

MPAS-O

Model for Prediction Across Scales

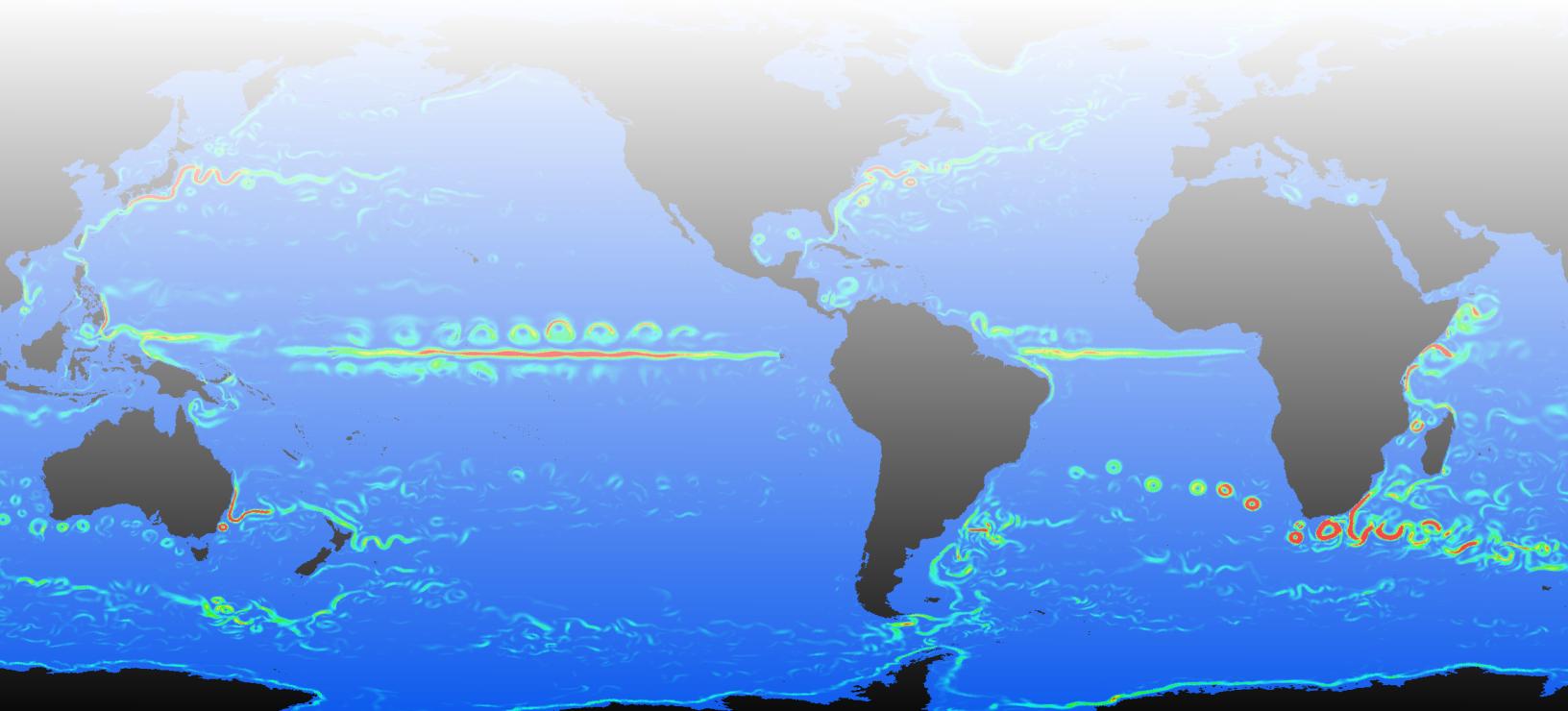
MPAS-Ocean Model User's Guide

Version: 1.0

Climate, Ocean, Sea-Ice Modeling Team

Los Alamos National Laboratory

June 14, 2013



Copyright

Copyright ©2013, Los Alamos National Security, LLC (LANS) (LA-CC-13-047) and the University Corporation for Atmospheric Research (UCAR).

All rights reserved.

LANS is the operator of the Los Alamos National Laboratory under Contract No. DE-AC52-06NA25396 with the U.S. Department of Energy. UCAR manages the National Center for Atmospheric Research under Cooperative Agreement ATM-0753581 with the National Science Foundation. The U.S. Government has rights to use, reproduce, and distribute this software. NO WARRANTY, EXPRESS OR IMPLIED IS OFFERED BY LANS, UCAR OR THE GOVERNMENT AND NONE OF THEM ASSUME ANY LIABILITY FOR THE USE OF THIS SOFTWARE. If software is modified to produce derivative works, such modified software should be clearly marked, so as not to confuse it with the version available from LANS and UCAR.

Additionally, redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- 1) Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- 2) Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- 3) None of the names of LANS, UCAR or the names of its contributors, if any, may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Foreword

The Model for Prediction Across Scales-Ocean (MPAS-Ocean) is an unstructured-mesh ocean model capable of using enhanced horizontal resolution in selected regions of the ocean domain. Model domains may be spherical with bottom topography to simulate the earth's oceans, or on Cartesian domains for idealized experiments. The global meshes, created using Spherical Centroidal Voronoi Tesselations (Ringler et al., 2008, 2011) consist of gridcells that vary smoothly from low to high resolution regions. Numerical algorithms specifically designed for these grids guarantee that mass, tracers, potential vorticity (in isopycnal mode) and energy are conserved (Thuburn et al., 2009; Ringler et al., 2010). MPAS-Ocean high-resolution and variable-resolution global simulations, as well as descriptions of mesh generation, model capabilities, and algorithms, are presented in Ringler et al. (2013).

MPAS-Ocean is one component within the MPAS framework of climate models that is developed in cooperation between Los Alamos National Laboratory (LANL) and the National Center for Atmospheric Research (NCAR). Functionality that is required by all cores, such as i/o, time management, block decomposition, etc, is developed collaboratively, and this code is shared across cores within the same repository. Each core then solves its own differential equations and physical parameterizations within this framework. This user's guide reflects the spirit of this collaborative process, where Part I, "The MPAS Framework", applies to all cores, and the remaining parts apply to MPAS-Ocean.

Here we would normally describe the new features of this version. Instead, for this initial release we describe those features that users will want but are not included in this public release. Most importantly, **this release does not include the ability to create or modify ocean grids or ocean initial conditions**. We appreciate that such functionality is important and we intend to release this capability when it meets our software standards. In addition, software interfaces to coupled MPAS-O into a coupled system model, such at the Community Earth System Model, is not contained in this release. And lastly, the suite of analysis tools available to process, interpret and visualize model output is limited. In summary, this release is intended help introduce MPAS-O to the scientific community with the expectation of expanding model's capability in future releases.

Information about MPAS-Ocean, including the most recent code, user's guide, and test cases, may be found at <http://mpas-dev.github.com>. This user's guide refers to version 1.0, with corresponding downloads at [release 1.0](#).

Contributors to this guide:

Doug Jacobsen, Mark Petersen, Todd Ringler

Additional contributors to MPAS Framework sections:

Michael Duda

Funding for the development of MPAS-Ocean was provided by the United States Department of Energy, Office of Science.

LA-UR-13-24348

History

A history of MPAS-O releases follows:

version	date	description
1.0	June 14th, 2013	Version 1.0 public release
0.0	June 14th, 2013	Initial pre-release of MPAS

Contents

1	MPAS-Ocean Quick Start Guide	13
I	The MPAS Framework	14
2	Building MPAS	15
2.1	Prerequisites	15
2.2	Compiling I/O Libraries	15
2.2.1	netCDF	15
2.2.2	parallel-netCDF	16
2.2.3	PIO	16
2.3	Compiling MPAS	16
2.4	Cleaning	18
2.5	Graph partitioning with METIS	18
3	Grid Description	20
4	Visualization	24
4.1	ParaView	24
II	MPAS-Ocean	27
5	Governing Equations	28
6	Dimensions	30
7	Namelist options	31
7.1	time_management	31
7.2	io	31
7.3	time_integration	32
7.4	grid	32
7.5	decomposition	33
7.6	hmix	33
7.7	hmix_del2	34
7.8	hmix_del4	35
7.9	hmix_Leith	35
7.10	standard_GM	36

7.11	Rayleigh_damping	36
7.12	vmix	36
7.13	vmix_const	37
7.14	vmix_rich	37
7.15	vmix_tanh	38
7.16	forcing	39
7.17	advection	39
7.18	bottom_drag	40
7.19	pressure_gradient	40
7.20	eos	41
7.21	eos_linear	41
7.22	split_explicit_ts	41
7.23	debug	42
8	Variable definitions	44
8.1	state	44
8.2	mesh	46
8.3	tend	48
8.4	diagnostics	49
8.5	scratch	49
9	Test Cases	50
9.1	Baroclinic Channel	50
9.1.1	Provided Files	50
9.1.2	Results	51
9.2	Overflow	53
9.2.1	Provided Files	53
9.2.2	Results	53
9.3	Real World Configuration	56
9.3.1	Provided Files	56
9.3.2	Results	56
10	Global Statistics	57
11	Ocean Visualization	59
11.1	Python	59
12	Running MPAS-Ocean within the CESM	60
13	Troubleshooting	61
13.1	Choice of time step	61
14	Known Issues	62

III Bibliography	63
IV Appendices	66
A Namelist options	67
A.1 time_management	67
A.1.1 config_do_restart	67
A.1.2 config_start_time	67
A.1.3 config_stop_time	67
A.1.4 config_run_duration	68
A.1.5 config_calendar_type	68
A.2 io	68
A.2.1 config_input_name	68
A.2.2 config_output_name	69
A.2.3 config_restart_name	69
A.2.4 config_restart_interval	69
A.2.5 config_output_interval	69
A.2.6 config_stats_interval	70
A.2.7 config_write_stats_on_startup	70
A.2.8 config_write_output_on_startup	70
A.2.9 config_frames_per_outfile	70
A.2.10 config_pio_num_iotasks	71
A.2.11 config_pio_stride	71
A.3 time_integration	71
A.3.1 config_dt	71
A.3.2 config_time_integrator	71
A.4 grid	72
A.4.1 config_num_halos	72
A.4.2 config_vert_coord_movement	72
A.4.3 config_alter_ICs_for_pbc	72
A.4.4 config_min_pbc_fraction	73
A.4.5 config_check_ssh_consistency	73
A.5 decomposition	73
A.5.1 config_block_decomp_file_prefix	73
A.5.2 config_number_of_blocks	74
A.5.3 config_explicit_proc_decomp	74
A.5.4 config_proc_decomp_file_prefix	74
A.6 hmix	74
A.6.1 config_hmix_ScaleWithMesh	74
A.6.2 config_maxMeshDensity	75
A.6.3 config_apvm_scale_factor	75
A.7 hmix_del2	75
A.7.1 config_use_mom_del2	75
A.7.2 config_use_tracer_del2	76
A.7.3 config_mom_del2	76
A.7.4 config_tracer_del2	76
A.8 hmix_del4	76

A.8.1 config_use_mom_del4	76
A.8.2 config_use_tracer_del4	77
A.8.3 config_mom_del4	77
A.8.4 config_tracer_del4	77
A.9 hmix_Leith	77
A.9.1 config_use_Leith_del2	77
A.9.2 config_Leith_parameter	78
A.9.3 config_Leith_dx	78
A.9.4 config_Leith_visc2_max	78
A.10 standard_GM	78
A.10.1 config_h_kappa	78
A.10.2 config_h_kappa_q	79
A.11 Rayleigh_damping	79
A.11.1 config_Rayleigh_friction	79
A.11.2 config_Rayleigh_damping_coeff	79
A.12 vmix	80
A.12.1 config_convective_visc	80
A.12.2 config_convective_diff	80
A.13 vmix_const	80
A.13.1 config_use_const_visc	80
A.13.2 config_use_const_diff	80
A.13.3 config_vert_visc	81
A.13.4 config_vert_diff	81
A.14 vmix_rich	81
A.14.1 config_use_rich_visc	81
A.14.2 config_use_rich_diff	81
A.14.3 config_bkrd_vert_visc	82
A.14.4 config_bkrd_vert_diff	82
A.14.5 config_rich_mix	82
A.15 vmix_tanh	83
A.15.1 config_use_tanh_visc	83
A.15.2 config_use_tanh_diff	83
A.15.3 config_max_visc_tanh	83
A.15.4 config_min_visc_tanh	83
A.15.5 config_max_diff_tanh	84
A.15.6 config_min_diff_tanh	84
A.15.7 config_zMid_tanh	84
A.15.8 config_zWidth_tanh	84
A.16 forcing	85
A.16.1 config_use_monthly_forcing	85
A.16.2 config_restoreTS	85
A.16.3 config_restoreT_timescale	85
A.16.4 config_restoreS_timescale	85
A.17 advection	86
A.17.1 config_vert_tracer_adv	86
A.17.2 config_vert_tracer_adv_order	86
A.17.3 config_horiz_tracer_adv_order	86
A.17.4 config_coef_3rd_order	86

A.17.5 config_monotonic	87
A.18 bottom_drag	87
A.18.1 config_bottom_drag_coeff	87
A.19 pressure_gradient	87
A.19.1 config_pressure_gradient_type	87
A.19.2 config_density0	88
A.20 eos	88
A.20.1 config_eos_type	88
A.21 eos_linear	88
A.21.1 config_eos_linear_alpha	88
A.21.2 config_eos_linear_beta	89
A.21.3 config_eos_linear_Tref	89
A.21.4 config_eos_linear_Sref	89
A.21.5 config_eos_linear_densityref	89
A.22 split_explicit_ts	90
A.22.1 config_n_ts_iter	90
A.22.2 config_n_bcl_iter_beg	90
A.22.3 config_n_bcl_iter_mid	90
A.22.4 config_n_bcl_iter_end	90
A.22.5 config_n_btr_subcycles	91
A.22.6 config_n_btr_cor_iter	91
A.22.7 config_vel_correction	91
A.22.8 config_btr_subcycle_loop_factor	91
A.22.9 config_btr_gam1_velWt1	92
A.22.10 config_btr_gam2_SSHWt1	92
A.22.11 config_btr_gam3_velWt2	92
A.22.12 config_btr_solve_SSH2	92
A.23 debug	93
A.23.1 config_check_zlevel_consistency	93
A.23.2 config_filter_btr_mode	93
A.23.3 config_prescribe_velocity	93
A.23.4 config_prescribe_thickness	93
A.23.5 config_include_KE_vertex	94
A.23.6 config_check_tracer_monotonicity	94
A.23.7 config_disable_thick_all_tend	94
A.23.8 config_disable_thick_hadv	95
A.23.9 config_disable_thick_vadv	95
A.23.10 config_disable_vel_all_tend	95
A.23.11 config_disable_vel_coriolis	95
A.23.12 config_disable_vel_pgrad	96
A.23.13 config_disable_vel_hmix	96
A.23.14 config_disable_vel_windstress	96
A.23.15 config_disable_vel_vmix	96
A.23.16 config_disable_vel_vadv	97
A.23.17 config_disable_tr_all_tend	97
A.23.18 config_disable_tr_adv	97
A.23.19 config_disable_tr_hmix	97
A.23.20 config_disable_tr_vmix	98

B Variable definitions	99
B.1 state	99
B.1.1 temperature	99
B.1.2 salinity	99
B.1.3 xtime	100
B.1.4 normalVelocity	100
B.1.5 layerThickness	100
B.1.6 density	100
B.1.7 normalBarotropicVelocity	101
B.1.8 ssh	101
B.1.9 normalBarotropicVelocitySubcycle	101
B.1.10 sshSubcycle	102
B.1.11 barotropicThicknessFlux	102
B.1.12 barotropicForcing	102
B.1.13 normalBaroclinicVelocity	103
B.1.14 zMid	103
B.1.15 tangentialVelocity	103
B.1.16 uTransport	104
B.1.17 uBolusGM	104
B.1.18 uBolusGMX	104
B.1.19 uBolusGMY	104
B.1.20 uBolusGMZ	105
B.1.21 uBolusGMZonal	105
B.1.22 uBolusGMMeridional	105
B.1.23 hEddyFlux	106
B.1.24 h_kappa	106
B.1.25 h_kappa_q	106
B.1.26 divergence	107
B.1.27 relativeVorticity	107
B.1.28 relativeVorticityCell	107
B.1.29 normalizedRelativeVorticityEdge	108
B.1.30 normalizedPlanetaryVorticityEdge	108
B.1.31 normalizedRelativeVorticityCell	108
B.1.32 layerThicknessEdge	109
B.1.33 layerThicknessVertex	109
B.1.34 kineticEnergyCell	109
B.1.35 normalVelocityX	110
B.1.36 normalVelocityY	110
B.1.37 normalVelocityZ	110
B.1.38 normalVelocityZonal	111
B.1.39 normalVelocityMeridional	111
B.1.40 montgomeryPotential	111
B.1.41 pressure	111
B.1.42 vertTransportVelocityTop	112
B.1.43 vertVelocityTop	112
B.1.44 displacedDensity	112
B.1.45 BruntVaisalaFreqTop	113
B.1.46 viscosity	113

B.1.47	circulation	113
B.1.48	areaCellGlobal	114
B.1.49	areaEdgeGlobal	114
B.1.50	areaTriangleGlobal	114
B.1.51	volumeCellGlobal	115
B.1.52	volumeEdgeGlobal	115
B.1.53	CFLNumberGlobal	115
B.1.54	nAverage	116
B.1.55	avgSsh	116
B.1.56	varSsh	116
B.1.57	avgNormalVelocityZonal	116
B.1.58	avgNormalVelocityMeridional	117
B.1.59	varNormalVelocityZonal	117
B.1.60	varNormalVelocityMeridional	117
B.1.61	avgNormalVelocity	118
B.1.62	varNormalVelocity	118
B.1.63	avgVertVelocityTop	118
B.2	mesh	119
B.2.1	latCell	119
B.2.2	lonCell	119
B.2.3	xCell	119
B.2.4	yCell	119
B.2.5	zCell	120
B.2.6	indexToCellID	120
B.2.7	latEdge	120
B.2.8	lonEdge	121
B.2.9	xEdge	121
B.2.10	yEdge	121
B.2.11	zEdge	121
B.2.12	indexToEdgeID	122
B.2.13	latVertex	122
B.2.14	lonVertex	122
B.2.15	xVertex	122
B.2.16	yVertex	123
B.2.17	zVertex	123
B.2.18	indexToVertexID	123
B.2.19	meshDensity	124
B.2.20	meshScalingDel2	124
B.2.21	meshScalingDel4	124
B.2.22	meshScaling	124
B.2.23	cellsOnEdge	125
B.2.24	nEdgesOnCell	125
B.2.25	nEdgesOnEdge	125
B.2.26	edgesOnCell	126
B.2.27	edgesOnEdge	126
B.2.28	weightsOnEdge	126
B.2.29	dvEdge	127
B.2.30	dcEdge	127

B.2.31	angleEdge	127
B.2.32	areaCell	127
B.2.33	areaTriangle	128
B.2.34	edgeNormalVectors	128
B.2.35	localVerticalUnitVectors	128
B.2.36	cellTangentPlane	129
B.2.37	cellsOnCell	129
B.2.38	verticesOnCell	129
B.2.39	verticesOnEdge	129
B.2.40	edgesOnVertex	130
B.2.41	cellsOnVertex	130
B.2.42	kiteAreasOnVertex	130
B.2.43	fEdge	131
B.2.44	fVertex	131
B.2.45	bottomDepth	131
B.2.46	deriv_two	131
B.2.47	adv_coefs	132
B.2.48	adv_coefs_2nd	132
B.2.49	adv_coefs_3rd	132
B.2.50	advCellsForEdge	133
B.2.51	nAdvCellsForEdge	133
B.2.52	highOrderAdvectionMask	133
B.2.53	lowOrderAdvectionMask	133
B.2.54	defc_a	134
B.2.55	defc_b	134
B.2.56	coeffs_reconstruct	134
B.2.57	maxLevelCell	135
B.2.58	maxLevelEdgeTop	135
B.2.59	maxLevelEdgeBot	135
B.2.60	maxLevelVertexTop	136
B.2.61	maxLevelVertexBot	136
B.2.62	refBottomDepth	136
B.2.63	refBottomDepthTopOfCell	136
B.2.64	vertCoordMovementWeights	137
B.2.65	boundaryEdge	137
B.2.66	boundaryVertex	137
B.2.67	boundaryCell	138
B.2.68	edgeMask	138
B.2.69	vertexMask	138
B.2.70	cellMask	139
B.2.71	normalVelocityForcing	139
B.2.72	temperatureRestore	139
B.2.73	salinityRestore	139
B.2.74	windStressMonthly	140
B.2.75	temperatureRestoreMonthly	140
B.2.76	salinityRestoreMonthly	140
B.2.77	edgeSignOnCell	141
B.2.78	edgeSignOnVertex	141

B.2.79	kiteIndexOnCell	141
B.2.80	seaSurfacePressure	142
B.3	tend	142
B.3.1	tend_temperature	142
B.3.2	tend_salinity	142
B.3.3	tend_normalVelocity	143
B.3.4	tend_layerThickness	143
B.3.5	tend_ssh	143
B.4	diagnostics	143
B.4.1	RiTopOfCell	143
B.4.2	RiTopOfEdge	144
B.4.3	vertViscTopOfEdge	144
B.4.4	vertDiffTopOfCell	144
B.4.5	windStressZonal	145
B.4.6	windStressMeridional	145
B.5	scratch	145
B.5.1	normalVelocityForcingX	145
B.5.2	normalVelocityForcingY	146
B.5.3	normalVelocityForcingZ	146
B.5.4	normalVelocityForcingZonal	146
B.5.5	normalVelocityForcingMeridional	147
B.5.6	vorticityGradientTangentialComponent	147
B.5.7	vorticityGradientNormalComponent	147
B.5.8	normalizedRelativeVorticityVertex	148
B.5.9	normalizedPlanetaryVorticityVertex	148
B.5.10	kineticEnergyVertex	148
B.5.11	kineticEnergyVertexOnCells	148

Chapter 1

MPAS-Ocean Quick Start Guide

This chapter provides MPAS-Ocean users with a quick start description of how to build and run the model. It is meant merely as a brief overview of the process, while the more detailed descriptions of each step are provided in later sections.

In general, the build process follows the following steps.

1. Build MPI Layer (OpenMPI, MVAPICH2, etc.)
2. Build serial NetCDF library (v3.6.3, v4.1.3, etc.)
3. Build Parallel-NetCDF library (v1.2.1, v1.3.0, etc.)
4. Build Parallel I/O library (v1.4.1, v1.6.1, etc.)
5. (Optional) Build METIS library and executables (v4.0, v5.0.2, etc.)
6. Checkout MPAS-Ocean from repository
7. Build ocean core

After step 7, an executable should be created called `ocean_model.exe`. Once the executable is built, one can begin the run process as follows:

1. Create run directory.
2. Copy executable to run directory.
3. Copy `namelist.input`, `input` and `graph` files into run directory. See test cases, Chapter 9. The standard first test case is the 10km-resolution baroclinic channel, available at
http://mpas-dev.github.com/ocean/release_1.0/release_1.0.html
as `overflow_10km_40layer.tgz`.
4. Edit `namelist.input` to have the proper parameters.
If step 4 was skipped, ensure paths to `input` and `graph` files are appropriately set.
5. (Optional) Create `graph` files, using METIS executable (`pmetis` or `gmetis` depending on version). A `graph` file is required for each processor count you want to use. See Section 2.5
6. Run MPAS-Ocean (e.g. `mpirun -np 8 ocean_model`).
7. Visualize output file, and perform analysis. See Chapters 4 and 11.

Part I

The MPAS Framework

Chapter 2

Building MPAS

2.1 Prerequisites

To build MPAS, compatible C and Fortran compilers are required. Additionally, the MPAS software relies on the PIO parallel I/O library to read and write model fields, and the PIO library requires the standard netCDF library as well as the parallel-netCDF library from Argonne National Labs. All libraries must be compiled with the same compilers that will be used to build MPAS. Section 2.2 summarizes the basic procedure of installing the required I/O libraries for MPAS.

In order for the MPAS makefiles to find the PIO, parallel-netCDF, and netCDF include files and libraries, the environment variables PIO, PNETCDF, and NETCDF should be set to the root installation directories of the PIO, parallel-netCDF, and netCDF installations, respectively. Newer versions of the netCDF library use a separate Fortran interface library; the top-level MPAS Makefile attempts to add `-lncdf` to the linker flags, but some linkers require that `-lncdf` appear before `-lncdf`, in which case `-lncdf` will need to be manually added just before `-lncdf` in the specification of LIBS in the top-level Makefile.

An MPI installation such as MPICH or OpenMPI is also required, and there is no option to build a serial version of the MPAS executables. There is currently no support for shared-memory parallelism with OpenMP within the MPAS framework.

2.2 Compiling I/O Libraries

NOTE: It's important to note the MPAS Developers are not responsible for any of the libraries that are used within MPAS. Support for specific libraries should be taken up with the respective developer groups.

Although most recent versions of the I/O libraries should work, the most tested versions of these libraries are: netCDF 4.1.3, parallel-netCDF 1.3.1, and PIO 1.4.1. The netCDF and parallel-netCDF libraries must be installed before building PIO library.

All commands are presented for csh, and will not work if pasted into another shell. Please translate them to the appropriate commands in your shell.

2.2.1 netCDF

Version 4.1.3 of the netCDF library may be downloaded from http://www.unidata.ucar.edu/downloads/netcdf/netcdf-4_1_3/index.jsp. Assuming the gfortran and gcc compilers will be used, the following shell commands are generally sufficient to install netCDF.

```

> setenv FC gfortran
> setenv F77 gfortran
> setenv F90 gfortran
> setenv CC gcc
> ./configure --prefix=XXXXX --disable-dap --disable-netcdf-4 --disable-cxx
--disable-shared --enable-fortran
> make all check
> make install

```

Here, XXXXX should be replaced with the directory that will serve as the root installation directory for netCDF. *Before proceeding to compile PIO the NETCDF_PATH environment variable should be set to the netCDF root installation directory.*

Certain compilers require addition flags in the CPPFLAGS environment variable. Please refer to the netCDF installation instructions for these flags.

2.2.2 parallel-netCDF

Version 1.3.1 of the parallel-netCDF library may be downloaded from <https://trac.mcs.anl.gov/projects/parallel-netcdf/wiki/Download>. Assuming the gfortran and gcc compilers will be used, the following shell commands are generally sufficient to install parallel-netCDF.

```

> setenv MPIF90 mpif90
> setenv MPIF77 mpif90
> setenv MPICC mpicc
> ./configure --prefix=XXXXX
> make
> make install

```

Here, XXXXX should be replaced with the directory that will serve as the root installation directory for parallel-netCDF. *Before proceeding to compile PIO the PNEDCDF_PATH environment variable should be set to the parallel-netCDF root installation directory.*

2.2.3 PIO

Instructions for building PIO can be found at <http://www.cesm.ucar.edu/models/pio/>. Please refer to these instructions for building PIO.

After PIO is built, and installed the PIO environment variable needs to be defined to point at the directory PIO is installed into. Older versions of PIO cannot be installed, and the PIO environment variable needs to be set to the directory where PIO was built instead.

2.3 Compiling MPAS

Before compiling MPAS, the NETCDF, PNEDCDF, and PIO environment variables must be set to the library installation directories as described in the previous section. A CORE variable also needs to either be defined or passed in during the make process. If CORE is not specified, the build process will fail.

The MPAS code uses only the ‘make’ utility for compilation. Rather than employing a separate configuration step before building the code, all information about compilers, compiler flags, etc.,

is contained in the top-level `Makefile`; each supported combination of compilers (i.e., a configuration) is included in the `Makefile` as a separate make target, and the user selects among these configurations by running `make` with the name of a build target specified on the command-line, e.g.,

```
> make gfortran
```

to build the code using the GNU Fortran and C compilers. Some of the available targets are listed in the table below, and additional targets can be added by simply editing the `Makefile` in the top-level directory.

Target	Fortran compiler	C compiler	MPI wrappers
xlf	xlf90	xlc	mpxlf90 / mpcc
pgi	pgf90	pgcc	mpif90 / mpicc
ifort	ifort	gcc	mpif90 / mpicc
gfortran	gfortran	gcc	mpif90 / mpicc
g95	g95	gcc	mpif90 / mpicc

In order to get a more complete and up-to-date list of available targets, one can use the following command within the top-level of MPAS. **NOTE:** This command is known to not work with Mac OSX.

```
> make -rpn | sed -n -e '/^$/ { n ; /^[^ ]*:/p }' | sed "s/: *.*$/g"
```

The MPAS framework supports multiple *cores* — currently a shallow water model, an ocean model, a non-hydrostatic atmosphere model, and a non-hydrostatic atmosphere initialization core — so the build process must be told which core to build. This is done by either setting the environment variable `CORE` to the name of the model core to build, or by specifying the core to be built explicitly on the command-line when running `make`. For the shallow water core, for example, one may run either

```
> setenv CORE sw
> make gfortran
```

or

```
> make gfortran CORE=sw
```

If the `CORE` environment variable is set and a core is specified on the command-line, the command-line value takes precedence; if no core is specified, either on the command line or via the `CORE` environment variable, the build process will stop with an error message stating such. Assuming compilation is successful, the model executable, named `${CORE} .model.exe` (e.g., `sw.model.exe`), should be created in the `src/` subdirectory, and a symbolic link to the model executable should exist in the top-level MPAS directory.

In order to get a list of available cores, one can simply run the top-level `Makefile` without setting the `CORE` environment variable, or passing the core via the command-line. An example of the output from this can be seen below.

```

> make
( make error )
make[1]: Entering directory '/home/douglasj/Documents svn-mpas-model.cgd.ucar.edu/trunk/mpas'

Usage: make target CORE=[core] [options]

Example targets:
ifort
gfortran
xlf
pgi

Available Cores:
atmosphere
init_atmosphere
ocean
sw

Available Options:
DEBUG=true      - builds debug version. Default is optimized version.
USE_PAPI=true   - builds version using PAPI for timers. Default is off.
TAU=true        - builds version using TAU hooks for profiling. Default is off.

Ensure that NETCDF, PNETCDF, PIO, and PAPI (if USE_PAPI=true) are environment variables
that point to the absolute paths for the libraries.

***** ERROR *****
No CORE specified. Quitting.
***** ERROR *****

make[1]: Leaving directory '/home/douglasj/Documents svn-mpas-model.cgd.ucar.edu/trunk/mpas'

```

2.4 Cleaning

To remove all files that were created when the model was built, including the model executable itself, `make` may be run for the ‘clean’ target:

```
> make clean
```

As with compiling, the core to be cleaned is specified by the `CORE` environment variable, or by specifying a core explicitly on the command-line with `CORE=`.

2.5 Graph partitioning with METIS

Before MPAS can be run in parallel, a mesh decomposition file with an appropriate number of partitions (equal to the number of MPI tasks that will be used) is required in the run directory. A limited number of mesh decomposition files (`graph.info.part.*`) are provided with each test case. In order to create new mesh decomposition files for your desired number of partitions, begin with the provided `graph.info` file and partition with METIS software (<http://glaros.dtc.umn.edu/gkhome/views/metis>). The serial graph partitioning program, METIS (rather than ParMETIS or

hMETIS) should be sufficient for quickly partitioning any SCVT produced by the grid_gen mesh generator.

After installing METIS, a `graph.info` file may be partitioned into N partitions by running

```
> gmetis graph.info N
```

The resulting file, `graph.info.part.N`, can then be copied into the MPAS run directory before running the model with N MPI tasks.

Chapter 3

Grid Description

This chapter provides a brief introduction to the common types of grids used in the MPAS framework.

The MPAS grid system requires the definition of seven elements. These seven elements are composed of two types of *cells*, two types of *lines*, and three types of *points*. These elements are depicted in Figure 3.1 and defined in Table 3.1. These elements can be defined on either the plane or the surface of the sphere. The two types of cells form two meshes, a primal mesh composed of Voronoi regions and a dual mesh composed of Delaunay triangles. Each corner of a primal mesh cell is uniquely associated with the “center” of a dual mesh cell and vice versa. So we define the two mesh as either a primal mesh (composed of cells P_i) or a dual mesh (composed of cells D_v). The center of any primal mesh cell, P_i , is denoted by \mathbf{x}_i and the center of any the dual mesh cell, D_v , is denoted by \mathbf{x}_v . The boundary of a given primal mesh cell P_i is composed of the set of lines that connect the \mathbf{x}_v locations of associated dual mesh cells D_v . Similarly, the boundary of a given dual mesh cell D_v is composed of the set of lines that connect the \mathbf{x}_i locations of the associated primal mesh cells P_i .

As shown in Figure 3.1, a line segment that connects two primal mesh cell centers is uniquely associated with a line segment that connects two dual mesh cell centers. We assume that these two line segments cross and the point of intersection is labeled as \mathbf{x}_e . In addition, we assume that these two line segments are orthogonal as indicated in Figure 3.1. Each \mathbf{x}_e is associated with two distances: d_e measures the distance between the primal mesh cells sharing \mathbf{x}_e and l_e measures the distance between the dual mesh cells sharing \mathbf{x}_e .

Since the two line segments crossing at \mathbf{x}_e are orthogonal, these line segments form a convenient local coordinate system for each edge. At each \mathbf{x}_e location a unit vector \mathbf{n}_e is defined to be parallel to the line connecting primal mesh cells. A second unit vector \mathbf{t}_e is defined such that $\mathbf{t}_e = \mathbf{k} \times \mathbf{n}_e$.

In addition to these seven element types, we require the definition of *sets of elements*. In all, eight different types of sets are required and these are defined and explained in Table 3.2 and Figure 3.2. The notation is always of the form of, for example, $i \in CE(e)$, where the LHS indicates the type of element to be gathered (cells) based on the RHS relation to another type of element (edges).

Table 3.3 provides the names of all *elements* and all *sets of elements* as used in the MPAS framework. Elements appear twice in the table when described in the grid file in more than one way, e.g. points are described with both cartesian and latitude/longitude coordinates. An “ncdump -h” of any MPAS grid, output or restart file will contain all variable names shown in second column of Table 3.3.

Table 3.1: Definition of elements used to build the MPAS grid.

<i>Element</i>	<i>Type</i>	<i>Definition</i>
\mathbf{x}_i	point	location of center of primal-mesh cells
\mathbf{x}_v	point	location of center of dual-mesh cells
\mathbf{x}_e	point	location of edge points where velocity is defined
d_e	line segment	distance between neighboring \mathbf{x}_i locations
l_e	line segment	distance between neighboring \mathbf{x}_v locations
P_i	cell	a cell on the primal-mesh
D_v	cell	a cell on the dual-mesh

Table 3.2: Definition of element groups used to reference connections in the MPAS grid.
Examples are provided in Figure 3.2.

<i>Syntax</i>	<i>output</i>
$e \in EC(i)$	set of edges that define the boundary of P_i .
$e \in EV(v)$	set of edges that define the boundary of D_v .
$i \in CE(e)$	two primal-mesh cells that share edge e .
$i \in CV(v)$	set of primal-mesh cells that form the vertices of dual mesh cell D_v .
$v \in VE(e)$	the two dual-mesh cells that share edge e .
$v \in VI(i)$	the set of dual-mesh cells that form the vertices of primal-mesh cell P_i .
$e \in ECP(e)$	edges of cell pair meeting at edge e .
$e \in EVC(v, i)$	edge pair associated with vertex v and mesh cell i .

Table 3.3: Variable names used to describe a MPAS grid.

<i>Element</i>	<i>Name</i>	<i>Size</i>	<i>Comment</i>
\mathbf{x}_i	{x,y,z}Cell	nCells	cartesian location of \mathbf{x}_i
\mathbf{x}_i	{lon,lat}Cell	nCells	longitude and latitude of \mathbf{x}_i
\mathbf{x}_v	{x,y,z}Vertex	nVertices	cartesian location of \mathbf{x}_v
\mathbf{x}_v	{lon,lat}Vertex	nVertices	longitude and latitude of \mathbf{x}_v
\mathbf{x}_e	{x,y,z}Edge	nEdges	cartesian location of \mathbf{x}_e
\mathbf{x}_e	{lon,lat}Edge	nEdges	longitude and latitude of \mathbf{x}_e
d_e	dcEdge	nEdges	distance between \mathbf{x}_i locations
l_e	dvEdge	nEdges	distance between \mathbf{x}_v locations
$e \in EC(i)$	edgesOnCell	(nEdgesMax,nCells)	edges that define P_i .
$e \in EV(v)$	edgesOnVertex	(3,nCells)	edges that define D_v .
$i \in CE(e)$	cellsOnEdge	(2,nEdges)	primal-mesh cells that share edge e .
$i \in CV(v)$	cellsOnVertex	(3,nVertices)	primal-mesh cells that define D_v .
$v \in VE(e)$	verticesOnEdge	(2,nEdges)	dual-mesh cells that share edge e .
$v \in VI(i)$	verticesOnCell	(nEdgesMax,nCells)	vertices that define P_i .

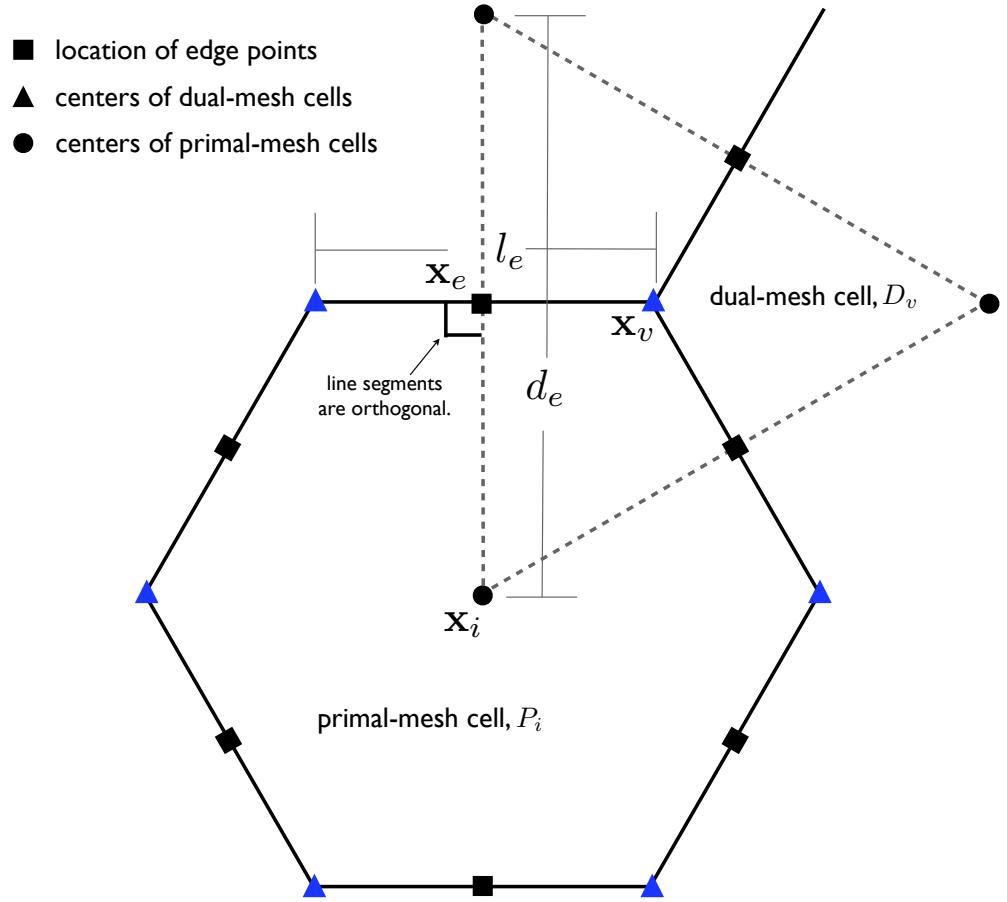


Figure 3.1: Definition of elements used to build the MPAS grid. Also see Table 3.1.

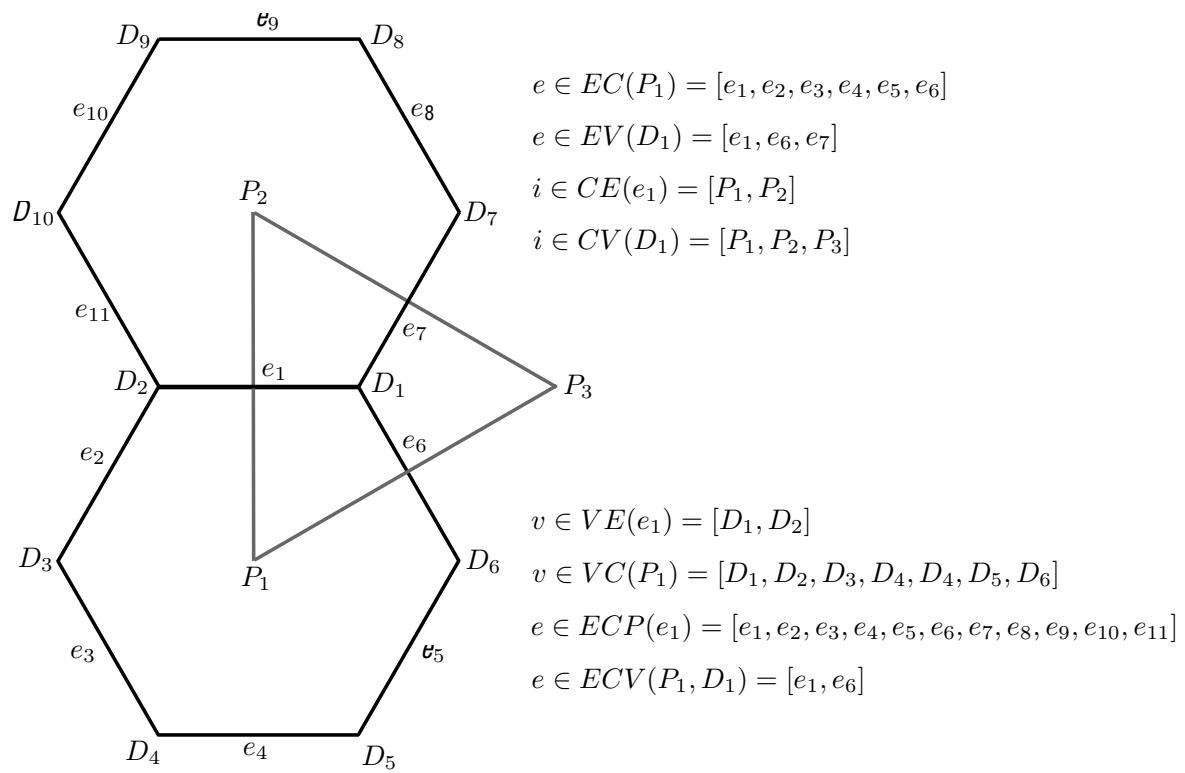


Figure 3.2: Definition of element groups used to reference connections in the MPAS grid.
Also see Table 3.2.

Chapter 4

Visualization

This chapter discusses visualization tools that may be used by all cores. For instructions on additional visualization tools for this core, see Chapter 11.

4.1 ParaView

ParaView may be used to visualize MPAS initialization, output, and restart files. It includes a reader that was specifically designed to read MPAS NetCDF files, including Cartesian and spherical domains. At this time, only cell-centered quantities may be plotted with ParaView. Variables located at edges and vertices must be interpolated to cell centers for visualization.

ParaView is freely available for download at <http://www.paraview.org>. Binary installations are available for Windows, Mac, and Linux, as well as source code files and tutorials. From the ParaView website:

ParaView is an open-source, multi-platform data analysis and visualization application. ParaView users can quickly build visualizations to analyze their data using qualitative and quantitative techniques. The data exploration can be done interactively in 3D or programmatically using ParaView's batch processing capabilities. ParaView was developed to analyze extremely large datasets using distributed memory computing resources. It can be run on supercomputers to analyze datasets of terascale as well as on laptops for smaller data.

To visualize an MPAS cell-centered variable in ParaView, open the file and choose **MPAS NetCDF (Unstructured)** as the file format. In the lower left Object Inspector panel, choose your variables of interest (Figure 4.1). For large data sets, loading fewer variables will result in less wait time. Options are available for latitude-longitude projections, vertical level, etc. Click the 'Apply' button to load the data set. In the toolbars at the top, choose the variable to plot from the pull-down menu, and 'Surface' for the type of visualization. The color bar button displays a color bar, and the color scale editor button allows the user to manually change the color bar type and extents. The Filters menu provides computational tools for interactive data manipulation. Movies, in avi format or as individual frames, may be conveniently created with the **Save Animation** tool in the File menu.

Paraview may be used to view the grid from any MPAS NetCDF file by choosing **Wireframe** or **Surface With Edges** from the visualization-type pull-down menu (Figure 4.2). This produces a view of the Delaunay triangulation, which is the dual mesh to the primal Voronoi cell grid (Figure

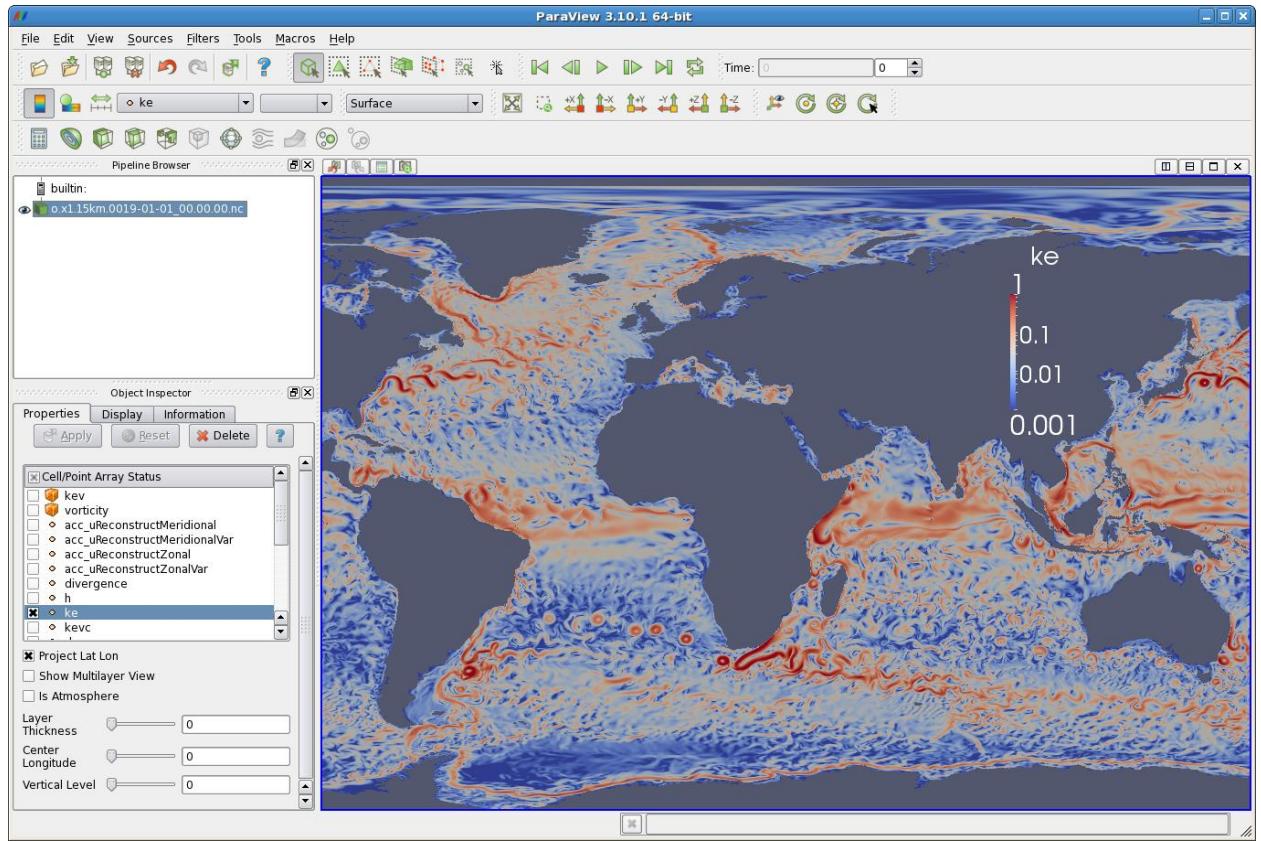


Figure 4.1: Example of ParaView to view an MPAS NetCDF file.

3.1). Paraview plots all variables by interpolating colors between each corner of the Delaunay triangles. These corners are the cell-center locations of the primal grid.

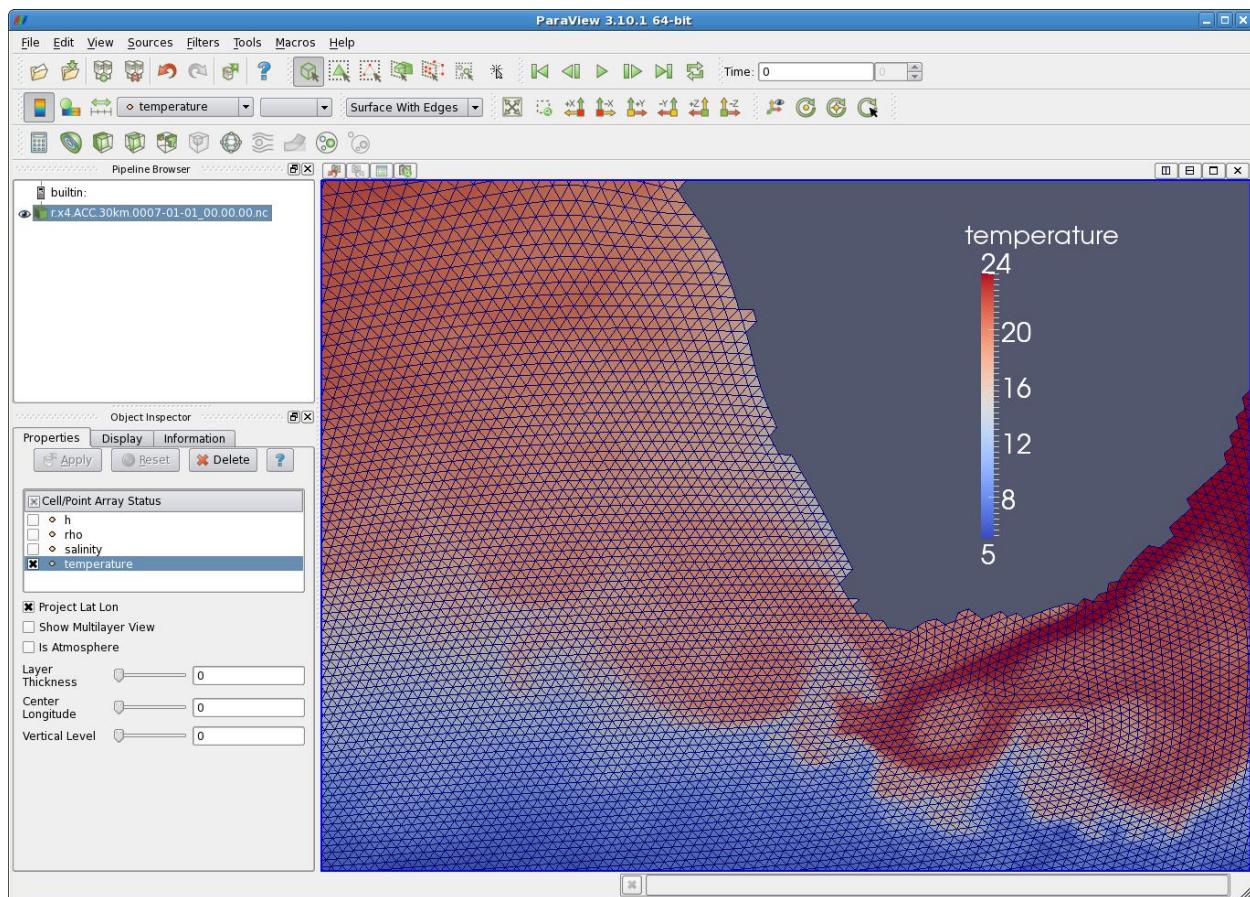


Figure 4.2: Example of visualizing the dual mesh from an MPAS NetCDF file.

Part II

MPAS-Ocean

Chapter 5

Governing Equations

The governing equations for MPAS-Ocean are

momentum equation:

$$\frac{\partial \mathbf{u}}{\partial t} + \eta \mathbf{k} \times \mathbf{u} + w \frac{\partial \mathbf{u}}{\partial z} = -\frac{1}{\rho_0} \nabla p - \frac{\rho g}{\rho_0} \nabla z^{mid} - \nabla K + \mathbf{D}_h^u + \mathbf{D}_v^u + \mathcal{F}^u \quad (5.1)$$

thickness equation:

$$\frac{\partial h}{\partial t} + \nabla \cdot (h \bar{\mathbf{u}}^z) + w|_{z=s^{top}} - w|_{z=s^{bot}} = 0 \quad (5.2)$$

tracer equation:

$$\frac{\partial}{\partial t} h \bar{\varphi}^z + \nabla \cdot (h \bar{\varphi} \bar{\mathbf{u}}^z) + \varphi w|_{z=s^{top}} - \varphi w|_{z=s^{bot}} = D_h^\varphi + D_v^\varphi + \mathcal{F}^\varphi \quad (5.3)$$

hydrostatic condition:

$$p(x, y, z) = p^s(x, y) + \int_z^{z^s} \rho g dz' \quad (5.4)$$

equation of state:

$$\rho = f_{eos}(\Theta, S, p) \quad (5.5)$$

Equations 5.1 through 5.5 are a normal expression of the primitive equations; i.e. the incompressible Boussinesq equations in hydrostatic balance. Variable definitions are in Tables 5.1 and 5.2. The momentum advection and Coriolis terms in (5.1) are presented in vorticity-kinetic energy form (Ringler et al., 2010, eqn 5). The thickness and tracer equations describe a single layer in the vertical, where the operator $\bar{(\cdot)}^z$ is a vertical average over that layer (see derivation in Appedix A.2 of Ringler et al. (2013)). Otherwise, 5.1–5.5 are the model equations in continuous form. Details of the conversion to fully discretized equations are given in the appendices of Ringler et al. (2013).

Table 5.1: Latin variables used in prognostic equation set. Column 3 shows the native horizontal grid location. All variables are located at the center of the layer in the vertical unless noted.

symbol	name	grid	notes
$\mathbf{D}_h^u, \mathbf{D}_v^u$	mom. diffusion terms	edge	h horizontal, v vertical
D_h^φ, D_v^φ	tracer diff. terms	cell	
f	Coriolis parameter	vertex	
f_{eos}	equation of state	-	
\mathcal{F}^u	momentum forcing	edge	
\mathcal{F}^φ	tracer forcing	cell	
g	grav. acceleration	constant	
h	layer thickness	cell	
\mathbf{k}	vertical unit vector		
K	kinetic energy	edge	$K = \mathbf{u} ^2 / 2$
p	pressure	cell	
p^s	surface pressure	cell	
q	potential vorticity	vertex	$q = \eta/h$
s^{bot}	z-location of bottom of layer	cell	
s^{top}	z-location of top of layer	cell	
S	salinity	cell	a tracer φ
t	time	-	
u	horizontal velocity	edge	normal component to edge
\mathbf{u}	horizontal velocity	-	
\mathbf{v}	3D velocity	-	
w	vertical transport	cell	top of layer in vertical
z	vertical coordinate	-	positive upward
z^{mid}	z-location of middle of layer	cell	
z^s	z-location of sea surface	cell	

Table 5.2: Greek variables used in prognostic equation set. Column 3 shows the native horizontal grid location. All variables are located at the center of the layer in the vertical.

symbol	name	grid	notes
δ	horizontal divergence	cell	$\delta = \nabla \cdot \mathbf{u}$
ζ	sea surface height	cell	
ω	relative vorticity	vertex	$\omega = \mathbf{k} \cdot (\nabla \times \mathbf{u})$
η	absolute vorticity	vertex	$\eta = \omega + f$
Θ	potential temperature	cell	a tracer φ
κ_h, κ_h	diffusion	cell	
ν_h, ν_v	viscosity	edge	
ρ	density	cell	
ρ_0	reference density	constant	
φ	generic tracer	cell	e.g. Θ, S

Chapter 6

Dimensions

Name	Units	Description
nCells	<i>unitless</i>	The number of polygons in the primary grid.
nEdges	<i>unitless</i>	The number of edge midpoints in either the primary or dual grid.
maxEdges	<i>unitless</i>	The largest number of edges any polygon within the grid has.
maxEdges2	<i>unitless</i>	Two times the largest number of edges any polygon within the grid has.
nAdvectionCells	<i>unitless</i>	The largest number of advection cells for any edge.
nVertices	<i>unitless</i>	The total number of cells in the dual grid. Also the number of corners in the primary grid.
TWO	<i>unitless</i>	The number two as a dimension.
R3	<i>unitless</i>	The number three as a dimension.
FIFTEEN	<i>unitless</i>	The number 15 as a dimension.
TWENTYONE	<i>unitless</i>	The number 21 as a dimension.
vertexDegree	<i>unitless</i>	The number of cells or edges touching each vertex.
nVertLevels	<i>unitless</i>	The number of levels in the vertical direction. All vertical levels share the same horizontal locations.
nVertLevelsP1	<i>unitless</i>	The number of interfaces in the vertical direction.
nMonths	<i>unitless</i>	The number of forcing slices in the monthly forcing fields.

Chapter 7

Namelist options

Embedded links point to more detailed namelist information in the appendix.

7.1 time_management

General time management is handled by the time_management namelist record. Included options handle time-related parts of MPAS, such as the calendar and if the simulation is a restart or not.

Users should use this record to specify the beginning time of the simulation, and either the duration or the end of the simulation. Only the end or the duration need to be specified as the other is derived within MPAS from the beginning time and other specified one.

If both the run duration and stop time are specified, run duration is used in place of stop time.

Name	Description
config_do_restart	Determines if the initial conditions should be read from a restart file, or an input file.
config_start_time	Timestamp describing the initial time of the simulation. If it is set to 'file', the initial time is read from restart_timestamp.
config_stop_time	Timestamp describing the final time of the simulation. If it is set to 'none' the final time is determined from config_start_time and config_run_duration.
config_run_duration	Timestamp describing the length of the simulation. If it is set to 'none' the duration is determined from config_start_time and config_stop_time. config_run_duration overrides inconsistent values of config_stop_time.
config_calendar_type	Selection of the type of calendar that should be used in the simulation.

7.2 io

The io namelist record provides options for modifications to the I/O system of MPAS. These include frequency, file name, and parallelization options.

Name	Description
------	-------------

<code>config_input_name</code>	The path to the input file for the simulation.
<code>config_output_name</code>	The template path and name to the output file from the simulation. A time stamp is prepended to the extension of the file (.nc).
<code>config_restart_name</code>	The template path and name to the restart file for the simulation. A time stamp is prepended to the extension of the file (.nc) both for input and output.
<code>config_restart_interval</code>	Timestamp determining how often a restart file should be written.
<code>config_output_interval</code>	Timestamp determining how often an output file should be written.
<code>config_stats_interval</code>	Timestamp determining how often a global statistics files should be written.
<code>config_write_stats_on_startup</code>	Logical flag determining if statistics files should be written prior to the first time step.
<code>config_write_output_on_startup</code>	Logical flag determining if an output file should be written prior to the first time step.
<code>config_frames_per_outfile</code>	Integer specifying how many time frames should be included in an output file. Once the maximum is reached, a new output file is created.
<code>config_pio_num_iotasks</code>	Integer specifying how many IO tasks should be used within the PIO library. A value of 0 causes all MPI tasks to also be IO tasks. IO tasks are required to write contiguous blocks of data to a file.
<code>config_pio_stride</code>	Integer specifying the stride of each IO task.

7.3 time_integration

The time integration namelist controls parameters that pertain to all time-stepping methods. Options that are specific to a particular time-stepping method are contained in a separate namelist for that method, below.

Name	Description
<code>config_dt</code>	Length of model time-step.
<code>config_time_integrator</code>	Time integration method.

7.4 grid

The grid namelist records provides options that modify both the horizontal and vertical MPAS-Ocean grids. This namelist record can be used to modify the number of halo cells each computational block contains, as well as changing how vertical coordinates move.

Name	Description
<code>config_num_halos</code>	Determines the number of halo cells extending from a blocks owned cells (Called the 0-Halo). The default of 3 is the minimum that can be used with monotonic advection.

<code>config_vert_coord_movement</code>	Determines the vertical coordinate movement type. 'uniform_stretching' distributes SSH perturbations through all vertical levels, 'fixed' places them all in the top level, 'user_specified' allows the input file to determine the distribution, and 'isopycnal' causes levels to be pure isopycnal.
<code>config_alter_ICs_for_pbcs</code>	Determines the method of alteration for partial bottom cells. 'zlevel_pbcs_on' alters the initial conditions for partial bottom cells, 'zlevel_pbcs_off' alters the initial conditions to have full cells everywhere, and 'off' does nothing to the initial conditions.
<code>config_min_pbc_fraction</code>	Determines the minimum fraction of a cell altering the initial conditions can create.
<code>config_check_ssh_consistency</code>	Enables a check to determine if the SSH is consistent across relevant variables.

7.5 decomposition

MPAS handles decomposing all variables into computational blocks. The decomposition used needs to be specified at run time and is computed by an external tool (e.g. metis). Additionally, MPAS supports multiple computational blocks per MPI process, and the user may specify an additional decomposition file which can specify the assignment of blocks to MPI processes. Run-time parameters that control the run-time decomposition used are specified within the decomposition namelist record.

Name	Description
<code>config_block_decomp_file_prefix</code>	Defines the prefix for the block decomposition file. Can include a path. The number of blocks is appended to the end of the prefix at run-time.
<code>config_number_of_blocks</code>	Determines the number of blocks a simulation should be run with. If it is set to 0, the number of blocks is the same as the number of MPI tasks at run-time.
<code>config_explicit_proc_decomp</code>	Determines if an explicit processor decomposition should be used. This is only useful if multiple blocks per processor are used.
<code>config_proc_decomp_file_prefix</code>	Defines the prefix for the processor decomposition file. This file is only read if <code>config_explicit_proc_decomp</code> is .true. The number of processors is appended to the end of the prefix at run-time.

7.6 hmix

There are several choices of horizontal mixing schemes available for the momentum and tracer equations. Each of these is a turbulence closure, and attempts to account for subgrid-scale mixing and diffusion. These schemes have the practical effect of reducing grid-scale noise in the velocity and tracer fields, and improving numerical stability.

Each horizontal mixing scheme has its own namelist, and may be turned on with the `_use_` logical configuration flags. Multiple schemes may be run simultaneously. The horizontal mixing

terms in the governing equations (5.1, 5.3) are \mathbf{D}_h^u for momentum and D_h^φ for tracers. No horizontal mixing is applied to the thickness equation.

All horizontal mixing coefficients can be set to scale with the mesh as $\rho_m^{-3/4}$ in equations (7.2, 7.3, 7.4, 7.5). The mesh density, ρ_m , is a variable in the input and restart file. It can vary between zero and one, and is one in the highest resolution region. Scaling with the mesh can be turned off, as described in the options below.

The anticipated potential Vorticity (APV) method is a parameterization of the effects of subgrid or unresolved scales on those explicitly resolved (Vallis and Hua, 1988). It contributes an upstream bias to the vorticity in the del2 and del4 momentum terms as follows,

$$\eta_{apv} = \eta - c_{apv} dt (\mathbf{u} \cdot \nabla \eta), \quad (7.1)$$

where the altered vorticity η_{apv} is used in equations (7.2, 7.8, 7.11).

Name	Description
config_hmix_ScaleWithMesh	If false, del2 and del4 coefficients are constant throughout the mesh (equivalent to setting $\rho_m = 1$ throughout the mesh). If true, these coefficients scale as mesh density to the -3/4 power.
config_maxMeshDensity	Global maximum of the mesh density
config_apvm_scale_factor	Anticipated potential vorticity (APV) method scale factor, c_{apv} . When zero, APV is off.

7.7 hmix_del2

The “del2”, or Laplacian, turbulence closures are

$$\mathbf{D}_h^u = \frac{\nu_h}{\rho_m^{3/4}} \nabla^2 \mathbf{u} = \frac{\nu_h}{\rho_m^{3/4}} (\nabla \delta + \mathbf{k} \times \nabla \eta), \quad (7.2)$$

$$D_h^\varphi = \nabla \cdot \left(h \frac{\kappa_h}{\rho_m^{3/4}} \nabla \varphi \right) \quad (7.3)$$

for momentum and tracers, respectively. Variable definitions appear in Tables 5.1 and 5.2. The momentum diffusion is in divergence-vorticity form because it is a natural discretization of the vector Laplacian operator with a C-grid staggering.

The Laplacian operator smooths the momentum and tracer fields, and smooths more strongly at small scales than at large scales. This operator is the two dimensional form of the heat equation, $u_t = \nu u_{xx}$, described in introductory books on partial differential equations. The strength of mixing is controlled by the viscosity, ν_h , for the momentum equation, and the diffusion, κ_h , for the tracer equation.

Name	Description
config_use_mom_del2	If true, Laplacian horizontal mixing is used on the momentum equation.
config_use_tracer_del2	If true, Laplacian horizontal mixing is used on the tracer equation.
config_mom_del2	Horizontal viscosity, ν_h .
config_tracer_del2	Horizontal diffusion, κ_h .

7.8 hmix_del4

The “del4”, or biharmonic, turbulence closures are

$$\mathbf{D}_h^u = -\frac{\nu_h}{\rho_m^{3/4}} \nabla^4 \mathbf{u} \quad (7.4)$$

$$D_h^\varphi = -\nabla \cdot \left(h \frac{\kappa_h}{\rho_m^{3/4}} \nabla [\nabla \cdot (h \nabla \varphi)] \right) \quad (7.5)$$

for momentum and tracers. These are both computed by applying the Laplacian operator twice. For momentum, this can be written in terms of divergence and vorticity as

$$\delta = \nabla \cdot \mathbf{u} \quad (7.6)$$

$$\eta = \mathbf{k} \cdot (\nabla \times \mathbf{u}) + f \quad (7.7)$$

$$\nabla^2 \mathbf{u} = (\nabla \delta + \mathbf{k} \times \nabla \eta) \quad (7.8)$$

$$\delta_2 = \nabla \cdot (\nabla^2 \mathbf{u}) \quad (7.9)$$

$$\eta_2 = \mathbf{k} \cdot (\nabla \times (\nabla^2 \mathbf{u})) + f \quad (7.10)$$

$$\mathbf{D}_h^u = \frac{\nu_h}{\rho_m^{3/4}} (\nabla \delta_2 + \mathbf{k} \times \nabla \eta_2). \quad (7.11)$$

The biharmonic operator is similar to the Laplacian operator, but smooths more strongly at high wavenumbers.

Name	Description
config_use_mom_del4	If true, biharmonic horizontal mixing is used on the momentum equation.
config_use_tracer_del4	If true, biharmonic horizontal mixing is used on the tracer equation.
config_mom_del4	Coefficient for horizontal biharmonic operator on momentum.
config_tracer_del4	Coefficient for horizontal biharmonic operator on tracers.

7.9 hmix_Leith

The Leith (1996) closure is the enstrophy-cascade analogy to the Smagorinsky (1963) energy-cascade closure, i.e. the Leith closure assumes an inertial range of enstrophy flux moving toward the grid scale. The assumption of an enstrophy cascade and dimensional analysis produces right-hand-side dissipation, \mathbf{D}_h^u , of velocity of the form

$$\mathbf{D}_h^u = \nabla \cdot (\nu_h \nabla \mathbf{u}) = \nabla \cdot \left(\Gamma |\nabla \omega| (\Delta x)^3 \nabla \mathbf{u} \right) \quad (7.12)$$

where ω is the relative vorticity, \mathbf{u} is the horizontal velocity, Δx is the local grid spacing and Γ is a non-dimensional, $O(1)$ parameter. This beta release approximates the RHS of the 7.12 as

$$\mathbf{D}_h^u = \nu_* \nabla_h^2 \mathbf{u} \quad (7.13)$$

where the $\nabla^2 \mathbf{u}$ is computed using the form shown in 7.8. Future releases will remove this approximation by computing the rate-of-strain, i.e. $\nabla \mathbf{u}$, directly.

Name	Description
config_use_Leith_del2	If true, the Leith enstrophy-cascade closure is turned on
config_Leith_parameter	Non-dimensional Leith closure parameter
config_Leith_dx	Characteristic length scale, usually the smallest dx in the mesh
config_Leith_visc2_max	Upper bound on the allowable value of Leith-computed viscosity

7.10 standard_GM

The Gent-McWilliams eddy parameterization is currently a stub, and is not available in the current release. Watch this space in a future release!

Name	Description
config_h_kappa	kappa parameter for Gent-McWilliams eddy parameterization
config_h_kappa_q	kappa-q parameter for Gent-McWilliams eddy parameterization

7.11 Rayleigh_damping

A linear damping toward a state of rest is available with this namelist option. It is implemented with a term on the RHS of the momentum equation (5.1) of the form

$$\mathcal{F}^u = -c_R \mathbf{u}. \quad (7.14)$$

Name	Description
config_Rayleigh_friction	If true, Rayleigh friction is included in the momentum equation.
config_Rayleigh_damping_coeff	Inverse-time coefficient for the Rayleigh damping term, c_R .

7.12 vmix

There are several choices of vertical mixing schemes available for the momentum and tracer equations. These are all applied with an implicit solve at the end of each time step using an operator splitting scheme.

Implicit vertical mixing is required in ocean models because vertical mixing between unstably stratified layers occurs at timescales faster than other model processes. The timestep requirement for explicit timestepping is usually set by the horizontal advective CFL condition. In order to include realistic vertical mixing without very small time steps, we use operator splitting so that the vertical momentum and tracer diffusion terms are treated with an implicit timestep, while the remaining terms of the momentum and tracer equations use an explicit method.

Each vertical mixing scheme has its own namelist, and may be turned on with the `_use_` logical configuration flags. Multiple schemes may be run simultaneously. The vertical mixing terms in the governing equations (5.1, 5.3) are

$$\mathbf{D}_v^u = \frac{\partial}{\partial z} \left(\nu_v \frac{\partial \mathbf{u}}{\partial z} \right), \quad (7.15)$$

$$D_v^\varphi = \rho \frac{\partial}{\partial z} \left(\kappa_v \frac{\partial \varphi}{\partial z} \right), \quad (7.16)$$

for momentum and tracers, respectively. No vertical mixing is applied to the thickness equation.

Name	Description
<code>config_convective_visc</code>	Value of vertical viscosity to be used in unstable conditions (i.e. Richardson number less than zero)
<code>config_convective_diff</code>	Value of vertical diffusion to be used in unstable conditions (i.e. Richardson number less than zero)

7.13 `vmix_const`

Here the vertical viscosity ν_v and diffusion κ_v are simply constant throughout the domain. This option is useful for testing and idealized domains but is not appropriate for real-world simulations, as the deep ocean should have much less mixing than the mixed layer.

Name	Description
<code>config_use_const_visc</code>	If true, constant vertical viscosity is included in the momentum equation
<code>config_use_const_diff</code>	If true, constant vertical diffusion is included in the tracer equation
<code>config_vert_visc</code>	Vertical viscosity, applied uniformly throughout domain
<code>config_vert_diff</code>	Vertical diffusion, applied uniformly throughout domain

7.14 `vmix_rich`

In the Richardson-number based vertical mixing parameterization (Pacanowski and Philander, 1981), the vertical diffusivity and viscosity are functions of the Richardson Number,

$$Ri = N^2 \left(\frac{\partial V}{\partial z} \right)^{-2} = -\frac{g}{\rho_0} \frac{\partial \rho}{\partial z} \left(\frac{\partial V}{\partial z} \right)^{-2}, \quad (7.17)$$

where $V = \sqrt{u^2 + v^2} = \sqrt{2ke}$ is the velocity magnitude. The discrete version is

$$Ri_k^{top} = -\frac{g}{\rho_0} \frac{\rho_{k-1}^* - \rho_k^*}{\frac{1}{2}(h_{k-1} + h_k)} \left(\frac{u_{k-1} - u_k}{\frac{1}{2}(h_{k-1} + h_k)} \right)^{-2} \quad (7.18)$$

$$= -\frac{g}{\rho_0} \frac{(\rho_{k-1}^* - \rho_k^*) \frac{1}{2}(h_{k-1} + h_k)}{(u_{k-1} - u_k)^2 + \epsilon} \quad (7.19)$$

where top indicates a layer interface, ke is the cell-centered kinetic energy, ρ_k^* is the density in layer k adiabatically displaced to the surface, and ϵ is a small number to avoid dividing by zero.

The variable Ri^{top} must be available at cell edges for the viscosity ν_v and at cell centers for the tracer diffusion κ_v . In addition, the computation of shear is native to the edges, while density is native to the cell centers.

The functional forms for vertical viscosity and diffusivity at each layer interface are as follows,

$$\nu_v = \nu_{bkrd} + c_{Ri}/(1 + 5Ri)^2 \quad (7.20)$$

$$\kappa_v = \kappa_{bkrd} + (\nu_{bkrd} + c_{Ri}/(1 + 5Ri)^2)/(1 + 5Ri) \quad (7.21)$$

for $Ri \geq 0$. For unstable stratification, $Ri < 0$ and the viscosity and diffusion are set to the convective values, which are typically very high.

Name	Description
config_use_rich_visc	If true, Richardson-number based vertical viscosity is included in the momentum equation
config_use_rich_diff	If true, Richardson-number based vertical diffusion is included in the tracer equation
config_bkrd_vert_visc	Background vertical viscosity for Richardson-number based vertical mixing, ν_{bkrd}
config_bkrd_vert_diff	Background vertical diffusion for Richardson-number based vertical mixing, κ_{bkrd}
config_rich_mix	Coefficient for Richardson-number vertical mixing function, c_{Ri}

7.15 [vmix_tanh](#)

Here the vertical viscosity ν_v and diffusion κ_v are produced with a hyperbolic tangent function, which produces more mixing in shallow waters and less in the deep ocean. It is uniform horizontally and in time.

Name	Description
config_use_tanh_visc	If true, tanh-based vertical viscosity is included in the momentum equation
config_use_tanh_diff	If true, tanh-based vertical diffusion is included in the tracer equation
config_max_visc_tanh	maximum viscosity value for tanh-fit function
config_min_visc_tanh	minimum viscosity value for tanh-fit function
config_max_diff_tanh	maximum diffusion value for tanh-fit function
config_min_diff_tanh	minimum diffusion value for tanh-fit function
config_zMid_tanh	z-coordinate location of center of tanh function
config_zWidth_tanh	vertical width parameter for tanh function

7.16 forcing

Forcing may be applied to the RHS of the momentum equation (5.1) through the term

$$\mathcal{F}^u = \frac{1}{\rho_0 h} \tau \quad (7.22)$$

where τ is typically the wind stress in N/m^2 applied to the top layer. More generally, momentum forcing may be applied to any layer in the ocean. The momentum forcing may be constant or monthly, as described in the configuration settings below. When running within the CESM, the wind stress is provided by the coupler (see Chapter 12).

Temperature and salinity restoring are applied to the tracer equation (5.3) through the term

$$\mathcal{F}^\varphi = -h \frac{\varphi - \varphi_r}{\tau_r} \quad (7.23)$$

where φ_r is the tracer restoring value and τ_r is the restoring timescale. This term is only applied at the top layer, and may be constant or monthly, as described in the configuration settings below. When running within the CESM, the coupler provides surface heat, salinity, and freshwater fluxes rather than a restoring in this form (see Chapter 12).

Name	Description
config_use_monthly_forcing	Controls time frequency of forcing. If false, a constant forcing is used, provided by the input fields normalVelocityForcing, temperatureRestore, and salinityRestore. If true, forcing is interpolated between monthly fields given by windStressMonthly, temperatureRestoreMonthly, and salinityRestoreMonthly.
config_restoreTS	If true, the restoring term is activated in the tracer equation for temperature and salinity.
config_restoreT_timescale	Restoring timescale for temperature, τ_r .
config_restoreS_timescale	Restoring timescale for salinity, τ_r .

7.17 advection

Three-dimensional tracer advection can be computed using 2^{nd} , 3^{rd} or 4^{th} flux reconstructions in the horizontal and vertical. In the horizontal, the high-order (i.e. 3^{rd} or 4^{th}) flux reconstruction is done following Skamarock and Gassmann (2011). Typically, the scheme is implemented with an upwind-bias ($\beta=0.25$ in (11) from Skamarock and Gassmann (2011)) to produce a 3^{rd} -order accurate reconstruction of tracer flux divergence on uniform hexagonal meshes. In the vertical, high-order estimates of tracer values at layer edges are reconstructed using a cubic spline. Monotone transport is guaranteed by blending these high-order flux approximations with the 1^{st} -order, upstream flux using the Zalesak (1979) flux-corrected transport scheme.

Name	Description
config_vert_tracer_adv	Method for interpolating tracer values from layer centers to layer edges
config_vert_tracer_adv_order	Order of polynomial used for tracer reconstruction at layer edges
config_horiz_tracer_adv_order	Order of polynomial used for tracer reconstruction at cell edges

<code>config_coef_3rd_order</code>	Reconstruction of 3rd-order reconstruction to blend with 4th-order reconstruction
<code>config_monotonic</code>	If .true. then fluxes are limited to produce a monotonic advection scheme

7.18 bottom_drag

The bottom drag is applied as a bottom boundary condition within the implicit solve of vertical mixing in the momentum equation (5.1), as

$$\lim_{z \rightarrow z_{bot}} \nu_v \frac{\partial u}{\partial z} = c_{drag} |u| u, \quad (7.24)$$

where c_{drag} is the dimensionless bottom drag coefficient, and z_{bot} is the z -location of the ocean bottom.

Name	Description
<code>config_bottom_drag_coeff</code>	Dimensionless bottom drag coefficient, c_{drag} .

7.19 pressure_gradient

For most choices of the vertical coordinate, the pressure gradient terms in the momentum equation will have the form

$$-\frac{1}{\rho_0} \nabla p - \frac{\rho g}{\rho_0} \nabla z^{mid}. \quad (7.25)$$

For isopycnal vertical coordinates, the user may choose to use the Montgomery potential,

$$M = \frac{1}{\rho} p + gz \quad (7.26)$$

and replace the pressure terms above with

$$-\nabla M. \quad (7.27)$$

See Higdon (2005, section 2.1) for details on the derivation and computation of the Montgomery potential.

Name	Description
<code>config_pressure_gradient_type</code>	Form of pressure gradient terms in momentum equation. For most applications, the gradient of pressure and layer mid-depth are appropriate. For isopycnal coordinates, one may use the gradient of the Montgomery potential.
<code>config_density0</code>	Density used as a coefficient of the pressure gradient terms, ρ_0 . This is a constant due to the Boussinesq approximation.

7.20 eos

Two forms of EOS are supported. The full EOS from Jackett and McDougall (1995) and a linear EOS.

Name	Description
config.eos.type	Character string to choose EOS formulation

7.21 eos_linear

The linear equation of state (leos) is specified as follows:

$$\rho = \rho_{ref} - \alpha_{leos}(T - T_{ref}) + \beta_{leos}(S - S_{ref}) \quad (7.28)$$

Name	Description
config.eos.linear_alpha	Linear thermal expansion coefficient
config.eos.linear_beta	Linear haline contraction coefficient
config.eos.linear_Tref	Reference temperature
config.eos.linear_Sref	Reference salinity
config.eos.linear_densityref	Reference density, i.e. density when T=Tref and S=Sref

7.22 split_explicit_ts

The split explicit time-stepping method solves the barotropic (vertically-integrated) velocities separately from the remaining baroclinic velocities. The time step for the barotropic solve is limited by fast surface gravity waves, and so is subcycled within a large timestep of the baroclinic velocity solve. This provides a 10 to 12-times speed-up over fourth-order Runge-Kutta time stepping.

A single large timestep in the split explicit algorithm may be summarized as

- Stage 1: solve for baroclinic velocity (3D)
- Stage 2: solve for barotropic velocity (2D) with explicit sub-cycling
- Stage 3: update thickness, tracers, density and pressure

The algorithm includes iterations within stage 1, within each subcycle of stage 2, and over the full three-stage process. Further details are provided in Ringler et al. (2013, Appendix A.5)

Name	Description
config.n_ts_iter	number of large iterations over stages 1-3
config.n_bcl_iter_beg	number of iterations of stage 1 (baroclinic solve) on the first split-explicit iteration

<code>config_n_bcl_iter_mid</code>	number of iterations of stage 1 (baroclinic solve) on any split-explicit iterations between first and last
<code>config_n_bcl_iter_end</code>	number of iterations of stage 1 (baroclinic solve) on the last split-explicit iteration
<code>config_n_btr_subcycles</code>	number of barotropic subcycles in stage 2
<code>config_n_btr_cor_iter</code>	number of iterations of the velocity corrector step in stage 2
<code>config_vel_correction</code>	If true, the velocity correction term is included in the horizontal advection of thickness and tracers
<code>config_btr_subcycle_loop_factor</code>	Barotropic subcycles proceed from t to $t + n\Delta t$, where n is this configuration option.
<code>config_btr_gam1_velWt1</code>	Weighting of velocity in the SSH predictor step in stage 2. When zero, previous subcycle time is used; when one, new subcycle time is used.
<code>config_btr_gam2_SSHWt1</code>	Weighting of SSH in the velocity corrector step in stage 2. When zero, previous subcycle time is used; when one, new subcycle time is used.
<code>config_btr_gam3_velWt2</code>	Weighting of velocity in the SSH corrector step in stage 2. When zero, previous subcycle time is used; when one, new subcycle time is used.
<code>config_btr_solve_SSH2</code>	If true, execute the SSH corrector step in stage 2

7.23 debug

At run-time a user can enable debugging features within MPAS-Ocean. These features include disabling any tendencies to help determine why an issue might be happening. Debugging options also include various checks on certain fields, and the ability to prescribe both a thickness and velocity field at run-time which are constant throughout a simulation. All options that control these debugging features are specified within the debug namelist record.

Name	Description
<code>config_check_zlevel_consistency</code>	Enables a run-time check for consistency for a zlevel grid. Ensures relevant variables correctly define the bottom of the ocean.
<code>config_filter_btr_mode</code>	Enables filtering of the barotropic mode.
<code>config_prescribe_velocity</code>	Enables a prescribed velocity field. This velocity field is read on input, and remains constant through a simulation.
<code>config_prescribe_thickness</code>	Enables a prescribed thickness field. This thickness field is read on input, and remains constant through a simulation.
<code>config_include_KE_vertex</code>	If true, the kinetic energy in each cell is computed by blending cell-based and vertex-based values of kinetic energy.
<code>config_check_tracer_monotonicity</code>	Enables a change on tracer monotonicity at the end of the monotonic advection routine. Only used if <code>config_monotonic</code> is set to .true.
<code>config_disable_thick_all_tend</code>	Disables all tendencies on the thickness field.
<code>config_disable_thick_hadv</code>	Disable tendencies on the thickness field from horizontal advection.
<code>config_disable_thick_vadv</code>	Disables tendencies on the thickness field from vertical advection.
<code>config_disable_vel_all_tend</code>	Disables all tendencies on the velocity field.
<code>config_disable_vel_coriolis</code>	Disables tendencies on the velocity field from the Coriolis force.
<code>config_disable_vel_pgrad</code>	Disables tendencies on the velocity field from the horizontal pressure gradient.

<code>config_disable_vel_hmix</code>	Disables tendencies on the velocity field from horizontal mixing.
<code>config_disable_vel_windstress</code>	Disables tendencies on the velocity field from horizontal wind stress.
<code>config_disable_vel_vmix</code>	Disables tendencies on the velocity field from vertical mixing.
<code>config_disable_vel_vadv</code>	Disables tendencies on the velocity field from vertical advection.
<code>config_disable_tr_all_tend</code>	Disables all tendencies on tracer fields.
<code>config_disable_tr_adv</code>	Disables tendencies on tracer fields from advection, both horizontal and vertical.
<code>config_disable_tr_hmix</code>	Disables tendencies on tracer fields from horizontal mixing.
<code>config_disable_tr_vmix</code>	Disables tendencies on tracer fields from vertical mixing.

Chapter 8

Variable definitions

Embedded links point to more detailed variable information in the appendix.

8.1 state

The state data structure contains a set of prognostic and diagnostic fields that are time dependent. The fields contained inside of state have two time levels available in the default version of MPAS-Ocean.

Name	Description
temperature	potential temperature
salinity	salinity
xtime	model time, with format 'YYYY-MM-DD_HH:MM:SS'
normalVelocity	horizontal velocity, normal component to an edge
layerThickness	layer thickness
density	density
normalBarotropicVelocity	barotropic velocity, used in split-explicit time-stepping
ssh	sea surface height
normalBarotropicVelocitySubcycle	barotropic velocity, used in subcycling in stage 2 of split-explicit time-stepping
sshSubcycle	sea surface height, used in subcycling in stage 2 of split-explicit time-stepping
barotropicThicknessFlux	Barotropic thickness flux at each edge, used to advance sea surface height in each subcycle of stage 2 of the split-explicit algorithm.
barotropicForcing	Barotropic tendency computed from the baroclinic equations in stage 1 of the split-explicit algorithm.
normalBaroclinicVelocity	baroclinic velocity, used in split-explicit time-stepping
zMid	z-coordinate of the mid-depth of the layer
tangentialVelocity	horizontal velocity, tangential to an edge
uTransport	horizontal velocity used to transport mass and tracers
uBolusGM	Bolus velocity in Gent-McWilliams eddy parameterization
uBolusGMX	Bolus velocity in Gent-McWilliams eddy parameterization, x-direction
uBolusGMY	Bolus velocity in Gent-McWilliams eddy parameterization, y-direction
uBolusGMZ	Bolus velocity in Gent-McWilliams eddy parameterization, z-direction

uBolusGMZonal	Bolus velocity in Gent-McWilliams eddy parameterization, zonal-direction
uBolusGMMeridional	Bolus velocity in Gent-McWilliams eddy parameterization, meridional-direction
hEddyFlux	Eddy flux in Gent-McWilliams eddy parameterization
h_kappa	kappa parameter for Gent-McWilliams eddy parameterization
h_kappa_q	kappa_q parameter for Gent-McWilliams eddy parameterization
divergence	divergence of horizontal velocity
relativeVorticity	curl of horizontal velocity, defined at vertices
relativeVorticityCell	curl of horizontal velocity, averaged from vertices to cell centers
normalizedRelativeVorticityEdge	curl of horizontal velocity divided by layer thickness, averaged from vertices to edges
normalizedPlanetaryVorticityEdge	earth's rotational rate (Coriolis parameter, f) divided by layer thickness, averaged from vertices to edges
normalizedRelativeVorticityCell	curl of horizontal velocity divided by layer thickness, averaged from vertices to cell centers
layerThicknessEdge	layer thickness averaged from cell center to edges
layerThicknessVertex	layer thickness averaged from cell center to vertices
kineticEnergyCell	kinetic energy of horizontal velocity on cells
normalVelocityX	component of horizontal velocity in the x-direction (cartesian)
normalVelocityY	component of horizontal velocity in the y-direction (cartesian)
normalVelocityZ	component of horizontal velocity in the z-direction (cartesian)
normalVelocityZonal	component of horizontal velocity in the eastward direction
normalVelocityMeridional	component of horizontal velocity in the northward
montgomeryPotential	Montgomery potential, may be used as the pressure for isopycnal coordinates.
pressure	pressure used in the momentum equation
vertTransportVelocityTop	vertical transport through the layer interface at the top of the cell
vertVelocityTop	vertical velocity defined at center (horizontally) and top (vertically) of cell
displacedDensity	potential density displaced to the mid-depth of top layer
BruntVaisalaFreqTop	Brunt Vaisala frequency defined at the center (horizontally) and top (vertically) of cell
viscosity	horizontal viscosity
circulation	area-integrated vorticity
areaCellGlobal	sum of the areaCell variable over the full domain, used to normalize global statistics
areaEdgeGlobal	sum of the areaEdge variable over the full domain, used to normalize global statistics
areaTriangleGlobal	sum of the areaTriangle variable over the full domain, used to normalize global statistics
volumeCellGlobal	sum of the volumeCell variable over the full domain, used to normalize global statistics
volumeEdgeGlobal	sum of the volumeEdge variable over the full domain, used to normalize global statistics
CFLNumberGlobal	maximum CFL number over the full domain
nAverage	number of timesteps in time-averaged variables
avgSsh	time-averaged sea surface height
varSsh	variance of sea surface height
avgNormalVelocityZonal	time-averaged velocity in the eastward direction
avgNormalVelocityMeridional	time-averaged velocity in the northward direction

<code>varNormalVelocityZonal</code>	variance of velocity in the eastward direction
<code>varNormalVelocityMeridional</code>	variance of velocity in the northward direction
<code>avgNormalVelocity</code>	time-averaged velocity, normal to cell edge
<code>varNormalVelocity</code>	variance of velocity, normal to cell edge
<code>avgVertVelocityTop</code>	time-averaged vertical velocity at top of cell

8.2 mesh

The mesh data type contains a single time level. The fields inside the mesh structure are not assumed to be time dependent. This data structure contains fields that describe the mesh, and the connectivity of the mesh. Several of the fields contained in this structure are shared throughout all MPAS dynamical cores.

Name	Description
<code>latCell</code>	Latitude location of cell centers in radians.
<code>lonCell</code>	Longitude location of cell centers in radians.
<code>xCell</code>	X Coordinate in cartesian space of cell centers.
<code>yCell</code>	Y Coordinate in cartesian space of cell centers.
<code>zCell</code>	Z Coordinate in cartesian space of cell centers.
<code>indexToCellID</code>	List of global cell IDs.
<code>latEdge</code>	Latitude location of edge midpoints in radians.
<code>lonEdge</code>	Longitude location of edge midpoints in radians.
<code>xEdge</code>	X Coordinate in cartesian space of edge midpoints.
<code>yEdge</code>	Y Coordinate in cartesian space of edge midpoints.
<code>zEdge</code>	Z Coordinate in cartesian space of edge midpoints.
<code>indexToEdgeID</code>	List of global edge IDs.
<code>latVertex</code>	Latitude location of vertices in radians.
<code>lonVertex</code>	Longitude location of vertices in radians.
<code>xVertex</code>	X Coordinate in cartesian space of vertices.
<code>yVertex</code>	Y Coordinate in cartesian space of vertices.
<code>zVertex</code>	Z Coordinate in cartesian space of vertices.
<code>indexToVertexID</code>	List of global vertex IDs.
<code>meshDensity</code>	Value of density function used to generate a particular mesh at cell centers.
<code>meshScalingDel2</code>	Coefficient to Laplacian mixing terms in momentum and tracer equations, so that viscosity and diffusion scale with mesh.
<code>meshScalingDel4</code>	Coefficient to biharmonic mixing terms in momentum and tracer equations, so that biharmonic viscosity and diffusion coefficients scale with mesh.
<code>meshScaling</code>	Coefficient used for mesh scaling, such as the Leith parameter.
<code>cellsOnEdge</code>	List of cells that straddle each edge.
<code>nEdgesOnCell</code>	Number of edges that border each cell.
<code>nEdgesOnEdge</code>	Number of edges that surround each of the cells that straddle each edge. These edges are used to reconstruct the tangential velocities.
<code>edgesOnCell</code>	List of edges that border each cell.
<code>edgesOnEdge</code>	List of edges that border each of the cells that straddle each edge.
<code>weightsOnEdge</code>	Reconstruction weights associated with each of the edgesOnEdge.
<code>dvEdge</code>	Length of each edge, computed as the distance between verticesOnEdge.

dcEdge	Length of each edge, computed as the distance between cellsOnEdge.
angleEdge	Angle the edge normal makes with local eastward direction.
areaCell	Area of each cell in the primary grid.
areaTriangle	Area of each cell (triangle) in the dual grid.
edgeNormalVectors	Normal vector defined at an edge.
localVerticalUnitVectors	Unit surface normal vectors defined at cell centers.
cellTangentPlane	The two vectors that define a tangent plane at a cell center.
cellsOnCell	List of cells that neighbor each cell.
verticesOnCell	List of vertices that border each cell.
verticesOnEdge	List of vertices that straddle each edge.
edgesOnVertex	List of edges that share a vertex as an endpoint.
cellsOnVertex	List of cells that share a vertex.
kiteAreasOnVertex	Area of the portions of each dual cell that are part of each cellsOn-Vertex.
fEdge	Coriolis parameter at edges.
fVertex	Coriolis parameter at vertices.
bottomDepth	Depth of the bottom of the ocean. Given as a positive distance from sea level.
deriv_two	Value of the second derivative of the polynomial used for reconstruction of cell center quantities at edges.
adv_coefs	Weighting coefficients used for reconstruction of cell center quantities at edges. Used in advection routines.
adv_coefs_2nd	Weighting coefficients used for reconstruction of cell center quantities at edges. Used in advection routines.
adv_coefs_3rd	Weighting coefficients used for reconstruction of cell center quantities at edges. Used in advection routines.
advCellsForEdge	List of cells used to reconstruct a cell quantity at an edge. Used in advection routines.
nAdvCellsForEdge	Number of cells used in reconstruction of cell center quantities at an edge. Used in advection routines.
highOrderAdvectionMask	Mask for high order advection. Values are 1 if high order is used, and 0 if not.
lowOrderAdvectionMask	Mask for low order advection. Values are 1 if low order is used, and 0 if not.
defc_a	Variable used with advection setup to compute advection coefficients. Deformation weight coefficients.
defc_b	Variable used with advection setup to compute advection coefficients. Deformation weight coefficients.
coeffs_reconstruct	Coefficients to reconstruct velocity vectors at cells centers.
maxLevelCell	Index to the last active ocean cell in each column.
maxLevelEdgeTop	Index to the last edge in a column with active ocean cells on both sides of it.
maxLevelEdgeBot	Index to the last edge in a column with at least one active ocean cell on either side of it.
maxLevelVertexTop	Index to the last vertex in a column with all active cells around it.
maxLevelVertexBot	Index to the last vertex in a column with at least one active ocean cell around it.
refBottomDepth	Reference depth of ocean for each vertical level. Used in 'z-level' type runs.
refBottomDepthTopOfCell	Reference depth of ocean for each vertical interface. Used in 'z-level' type runs.

<code>vertCoordMovementWeights</code>	Weights used for distribution of sea surface height perturbations through multiple vertical levels.
<code>boundaryEdge</code>	Mask for determining boundary edges. A boundary edge has only one active ocean cell neighboring it.
<code>boundaryVertex</code>	Mask for determining boundary vertices. A boundary vertex has at least one inactive cell neighboring it.
<code>boundaryCell</code>	Mask for determining boundary cells. A boundary cell has at least one inactive cell neighboring it.
<code>edgeMask</code>	Mask on edges that determines if computations should be done on edge.
<code>vertexMask</code>	Mask on vertices that determines if computations should be done on vertex.
<code>cellMask</code>	Mask on cells that determines if computations should be done on cell.
<code>normalVelocityForcing</code>	Velocity forcing field. Defines a forcing at an edge.
<code>temperatureRestore</code>	Temperature restoring field, for restoring temperature at the surface.
<code>salinityRestore</code>	Salinity restoring field, for restoring salinity at the surface.
<code>windStressMonthly</code>	Monthly wind stress field, defined at the surface for use in monthly forcing.
<code>temperatureRestoreMonthly</code>	Monthly temperature restoring field, defined at the surface for use in monthly forcing.
<code>salinityRestoreMonthly</code>	Monthly salinity restoring field, defined at the surface, for use in monthly forcing.
<code>edgeSignOnCell</code>	Sign of edge contributions to a cell for each edge on cell. Used for bit-reproducible loops. Represents directionality of vector connecting cells.
<code>edgeSignOnVertex</code>	Sign of edge contributions to a vertex for each edge on vertex. Used for bit-reproducible loops. Represents directionality of vector connecting vertices.
<code>kiteIndexOnCell</code>	Index of kite in dual grid, based on <code>verticesOnCell</code> .
<code>seaSurfacePressure</code>	Pressure defined at the sea surface.

8.3 tend

The tend data structure represents the tendencies used to time step the prognostic variables within the state structure.

Name	Description
<code>tend_temperature</code>	time tendency of potential temperature
<code>tend_salinity</code>	time tendency of salinity measured as change in practical salinity units per second
<code>tend_normalVelocity</code>	time tendency of normal component of velocity
<code>tend_layerThickness</code>	time tendency of layer thickness
<code>tend_ssh</code>	time tendency of sea-surface height

8.4 diagnostics

The diagnostics type contains a set of diagnostics variables that are only generally used in specific parts of MPAS-Ocean.

Name	Description
RiTopOfCell	gradient Richardson number defined at the center (horizontally) and top (vertically)
RiTopOfEdge	gradient Richardson number defined at the edge (horizontally) and top (vertically)
vertViscTopOfEdge	vertical viscosity defined at the edge (horizontally) and top (vertically)
vertDiffTopOfCell	vertical diffusion defined at the edge (horizontally) and top (vertically)
windStressZonal	reconstructed surface wind stress in the eastward direction.
windStressMeridional	reconstructed surface wind stress in the northward direction.

8.5 scratch

Name	Description
normalVelocityForcingX	reconstructed wind stress in the x-direction (cartesian)
normalVelocityForcingY	reconstructed wind stress in the y-direction (cartesian)
normalVelocityForcingZ	reconstructed wind stress in the z-direction (cartesian)
normalVelocityForcingZonal	reconstructed wind stress in the eastward direction
normalVelocityForcingMeridional	reconstructed wind stress in the northward direction
vorticityGradientTangentialComponent	gradient of vorticity in the tangent direction (positive points from vertex1 to vertex2)
vorticityGradientNormalComponent	gradient of vorticity in the normal direction (positive points from cell1 to cell2)
normalizedRelativeVorticityVertex	curl of horizontal velocity divided by layer thickness, defined at vertices
normalizedPlanetaryVorticityVertex	earth's rotational rate (Coriolis parameter, f) divided by layer thickness, defined at vertices
kineticEnergyVertex	kinetic energy of horizontal velocity defined at vertices
kineticEnergyVertexOnCells	kinetic energy of horizontal velocity defined at vertices

Chapter 9

Test Cases

All test cases can be downloaded from <http://mpas-dev.github.com>. The test cases for this version of the code are available at
http://mpas-dev.github.com/ocean/release_1.0/release_1.0..

9.1 Baroclinic Channel

This section describes a periodic channel, that is driven by a baroclinic instability generated via meridional temperature gradient. This test case comes from Ilicak et al. (2012). The domain is a planar channel that is periodic in the longitudinal direction with no-slip boundary conditions along the north and south boundaries. The longitudinal extent is 160 km while the latitudinal extent is 500 km. The vertical depth of the domain is 1000 m with a flat bottom. The channel is on a f -plane, with the Coriolis parameter $f = 1.2 \times 10^{-4} \text{ s}^{-1}$.

Temperature decreases downward and in the meridional direction. A cosine shape temperature perturbation with a wavelength 120 km in the zonal direction is used to instigate the baroclinic instability.

9.1.1 Provided Files

Three resolutions of the baroclinic channel are provided for exploration. First is a 10 km horizontal resolution, second is a 4 km horizontal resolution, and third is a 1 km horizontal resolution. All three horizontal resolutions have 20 vertical levels with uniform vertical resolution of 50 m.

Each resolution is provided in it's own tar file, which when extracted creates a "run directory" that is only missing the ocean_model.exe executable. Included in each tar file are the following files.

- grid.nc:

This is the input grid file that includes initial conditions.

It can be visualized in the same way output files can to see the initial conditions.

- graph.info:

This file is a graph of all of the cells in the mesh.

It is used to decompose the mesh into partition files.

- graph.info.part.2:

This is a partition file for use with a 2 processor run.

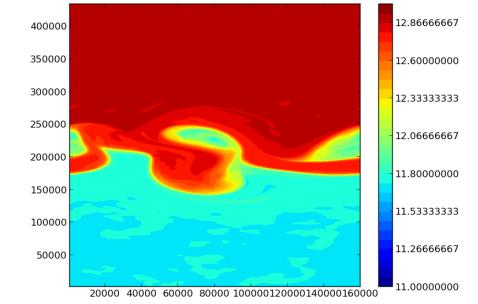
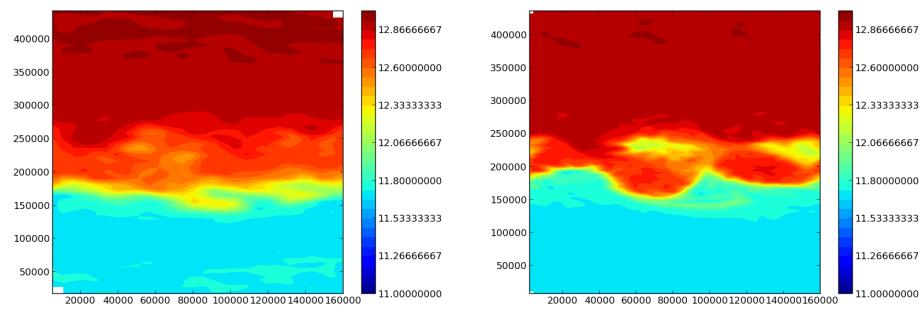
The graph.info file has been decomposed into two blocks for this file.

- `graph.info.part.4`:
This is a partition file for use with a 4 processor run.
The `graph.info` file has been decomposed into four blocks for this file.
- `graph.info.part.8`:
This is a partition file for use with a 8 processor run.
The `graph.info` file has been decomposed into eight blocks for this file.
- `namelist.input`:
This is the namelist file with all parameters for the run.
It has a default setup which when run provides the results in the next section.
- `visualize_channel.py`:
Python visualization script. See Section [11.1](#).

9.1.2 Results

The surface temperature fields at day 10 from the 10km, 4 km, and 1 km simulations are shown in Figure [9.1](#). This results should be reproduced using the default “run_directory” for the 10 km, 4 km, and 1 km resolutions. The python script used to visualize the output is included with the tar file, and described in Section [11.1](#).

Figure 9.1: Baroclinic channel test case results. Surface temperature field after 10 simulation days using default included namelists.



9.2 Overflow

The overflow test case is an idealized domain designed to investigate the impact of topography on spurious mixing, and is similar to those found in Haidvogel and Beckmann (1999) and Ilicak et al. (2012, section 4). The domain is on a Cartesian plane of regularly spaced hexagons in the horizontal. It is effectively two-dimensional, with dynamics occurring in the y-z plane, while the x-direction is four cells wide and periodic.

This test case is useful for parameter studies to compare resolution, vertical mixing schemes, strength and types of horizontal mixing, partial bottom cells, and choice of vertical coordinate. More advanced statistics, like the resting potential energy (Ilicak et al., 2012), may be used to produce quantitative assessments of these comparisons. The included test cases follow the parameters in Ilicak et al. (2012, section 4) and use zero explicit tracer mixing, Laplacian horizontal mixing of momentum with a viscosity of $10^3 \text{ m}^2/\text{s}$, and constant vertical mixing of momentum with a viscosity of $10^{-4} \text{ m}^2/\text{s}$. The equation of state is linear, and vertical coordinate is z-star (Adcroft and Campin, 2004).

9.2.1 Provided Files

- grid.nc:

This is the input grid file that includes initial conditions.

It can be visualized in the same way output files can to see the initial conditions.

- graph.info:

This file is a graph of all of the cells in the mesh.

It is used to decompose the mesh into partition files.

- graph.info.part.2:

This is a partition file for use with a 2 processor run.

The graph.info file has been decomposed into two blocks for this file.

- graph.info.part.4:

This is a partition file for use with a 4 processor run.

The graph.info file has been decomposed into four blocks for this file.

- graph.info.part.8:

This is a partition file for use with a 8 processor run.

The graph.info file has been decomposed into eight blocks for this file.

- namelist.input:

This is the namelist file with all parameters for the run.

It has a default setup which when run provides the results in the next section.

- visualize_overflow.py:

Python visualization script. See Section [11.1](#).

9.2.2 Results

Initially, a cold, dense volume of water is released at the top of a sill. Within the course of an 18-hour simulation the cold water descends the steep mount and continues northward along the bottom. The Coriolis parameter is set to zero so that rotational effects do not occur. The speed of descent, or equivalently the time to reach the bottom, is a simple way to measure the amount

of mixing that occurs as the plug of water descends. In the two cases included with this release, the cold front reaches the bottom of the sill after eight hours for the high resolution case (1 km horizontal mesh, 100 layers), but after 16 hours for the low resolution case (10 km horizontal mesh, 40 layers). Visualizations show that some ten-degree water remains after nine hours for the high-resolution case (Figure 9.2), but has all mixed out for the low-resolution case (not shown).

The python script `visualize_overflow.py` is included in the test case files to visualize cross-sections of cell-centered variables, like those shown in Figure 9.2. These plots may be created at the unix prompt with, for example,

```
python visualize_overflow.py -f output.nc -v temperature --min=10 --max=20 --time=9
```

The python scripts are further described in Section 11.1.

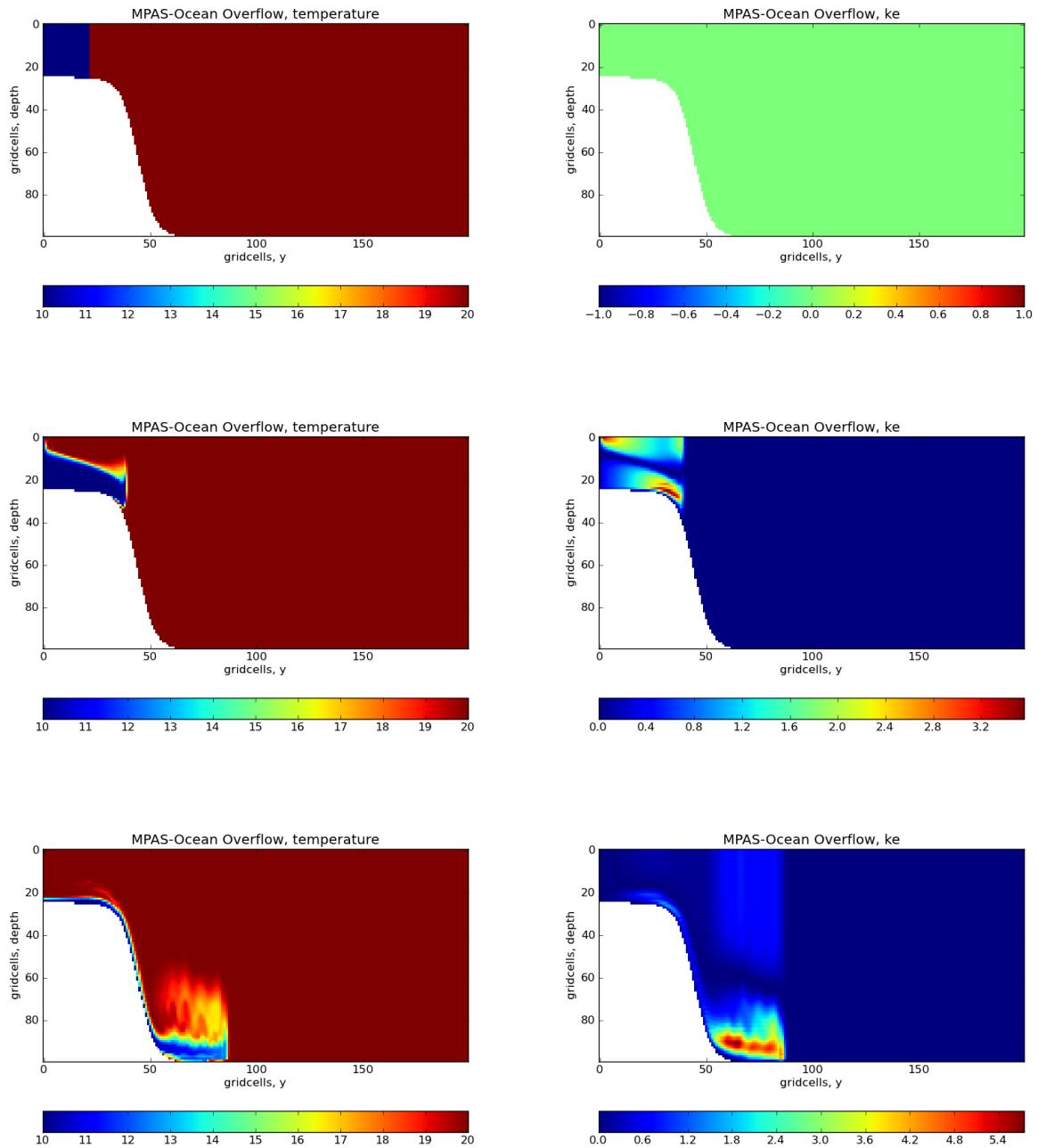


Figure 9.2: Overflow test case results for the high resolution case (1 km horizontal grid-cells), showing vertical cross-sections of temperature (left) and kinetic energy (right). Rows show initial condition (top), 3 hours (middle) and 9 hours (bottom).

9.3 Real World Configuration

describe test case here

9.3.1 Provided Files

- grid.nc:

This is the input grid file that includes initial conditions.

It can be visualized in the same way output files can to see the initial conditions.

- graph.info:

This file is a graph of all of the cells in the mesh.

It is used to decompose the mesh into partition files.

- graph.info.part.2:

This is a partition file for use with a 2 processor run.

The graph.info file has been decomposed into two blocks for this file.

- graph.info.part.4:

This is a partition file for use with a 4 processor run.

The graph.info file has been decomposed into four blocks for this file.

- graph.info.part.8:

This is a partition file for use with a 8 processor run.

The graph.info file has been decomposed into eight blocks for this file.

- namelist.input:

This is the namelist file with all parameters for the run.

It has a default setup which when run provides the results in the next section.

9.3.2 Results

results here

Chapter 10

Global Statistics

MPAS-Ocean simulations create global statistics text files that are accessible during the simulation, as described in Table 10.1. These are useful to check the health and progress of a simulation, and to compare the history of global statistics amongst similar simulations. Frequency is controlled by `_stats_` flags in the namelist. The files are simply text files formatted in columns and rows, where each row is a particular time, written to `stats_time.txt`, and each column is a variable, described in `stats_readme.txt`. Output is always appended to the `stats_*` files, so these files must be deleted or moved in order to restart from line one.

A chain of simple unix commands may be used to access a specific part of the data. For example, to view the last three values of column seven in the global average, use

```
cat stats_avg.txt | awk '{print $7}' | tail -n3
```

Gnuplot is a simple plotting tool available on Linux systems that can easily plot columns in text files. In this example, two MPAS-Ocean simulations were executed, in directories `run1` and `run2`. Begin Gnuplot with the `gnuplot` command at the unix prompt. Plot column seven from both runs using points with

```
plot 'run1/stats_avg.txt' u 7 w p, 'run2/stats_avg.txt' u 7 w p
```

and with lines using

```
plot 'run1/stats_avg.txt' u 7 w l, 'run2/stats_avg.txt' u 7 w l
```

Plot on a vertical log scale and add legend text with

```
set logscale y  
plot 'run1/stats_avg.txt' u 7 w l t 'run 1', 'run2/stats_avg.txt' u 7 w l t 'run 2'
```

It is often useful to view the difference between the global statistics of two simulations. This can be done with the `paste` command, which concatenates the two files horizontally. In this example, each file has 18 columns, so `$25` refers to column 7 of the second file:

```
plot '<paste run1/stats_avg.txt run2/stats_avg.txt' u ($7-$25)
```

Table 10.1: Global statistics output files.

file name	description
<code>stats_time.txt</code>	time stamp, rows correspond to rows of other stats files
<code>stats_readme.txt</code>	list of variables in columns of other stats files
<code>stats_min.txt</code>	minimum over the global domain
<code>stats_max.txt</code>	maximum over the global domain
<code>stats_sum.txt</code>	volume-weighted summation over the global domain
<code>stats_avg.txt</code>	volume-weighted average over the global domain
<code>stats_colmin.txt</code>	minimum column sum over the global domain
<code>stats_colmax.txt</code>	maximum column sum over the global domain

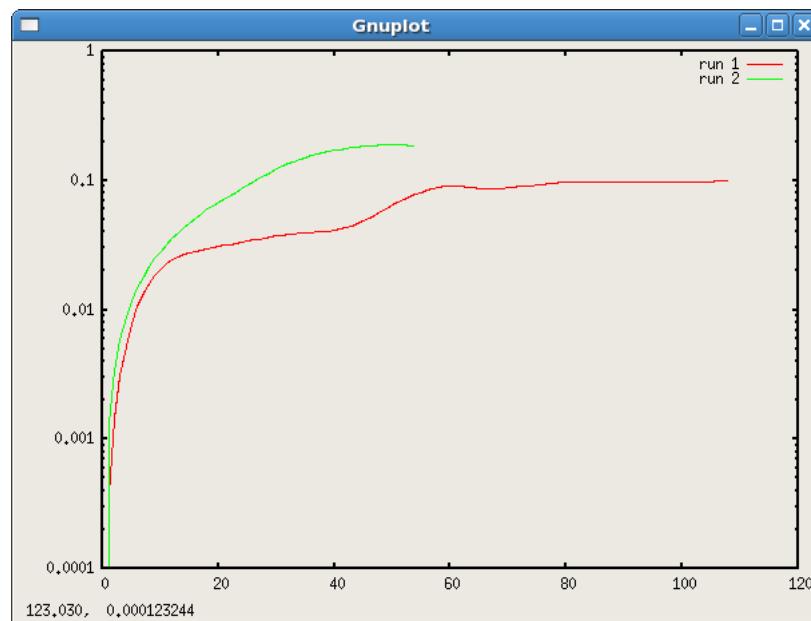


Figure 10.1: Example of plot of global statistics created with gnuplot.

Chapter 11

Ocean Visualization

This chapter discusses visualization tools that are specific to the ocean core. For instructions on visualization tools that may be used by all cores, such as Paraview, see Chapter 4.

11.1 Python

Python visualization scripts are provided with the tar files for the baroclinic channel and overflow test cases. In order to use these scripts, the following python modules are required:

- matplotlib, see <http://matplotlib.org>
- numpy, see <http://www.numpy.org>
- pylab, see www.scipy.org
- netCDF4, see <http://code.google.com/p/netcdf4-python>

A convenient way to install all these libraries at once is to purchase the Enthought Python Distribution (EPD), available at <https://www.enthought.com/products/epd>. Many institutions have Python-EPD installed on their compute clusters.

Examples of output from python visualization scripts are shown in Figures 9.1 and 9.2. Options for each script may be found using, e.g.

```
python visualize_overflow.py --help
```

These scripts are easily customizable by the user. Within the python script the specified variable is read from the NetCDF file using the commands

```
f = NetCDFFile(options.filename,'r')
field = f.variables[options.variable]
```

The plot is created with the matplotlib command `plt.imshow`, as described in the following tutorials:

- http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.imshow
- http://matplotlib.org/users/image_tutorial.html

Text is then added with the commands `xlabel`, `ylabel`, `title`; colorbar is added using `plt.colorbar`; and the figure is saved to the local directory as a png file using `plt.savefig`.

Chapter 12

Running MPAS-Ocean within the CESM

Running MPAS-Ocean within the CESM is currently a work in progress, but this chapter will be filled in later.

Chapter 13

Troubleshooting

13.1 Choice of time step

Symptoms: “Abort: NaN detected” appears in log.0000.err file.

Possible cause: Time step is too long.

Remedy: Shorten time step.

Discussion: The time step must be short enough that the CFL criterion is not violated. The minimum timestep varies based on resolution, viscosity, diffusion, forcing, and the resulting dynamics. The simulations presented in Ringler et al. (2013) used a 600s split explicit timestep for the global 15km mesh, and 360s for a global mesh with 7.5km regions. On a 120km global mesh, we typically use a 3000s for split explicit and 500s for fourth-order Runge-Kutta. These timesteps may need adjustment for your choice of turbulence closure and parameter values.

Chapter 14

Known Issues

- Plotting of periodic field with Paraview

Part III

Bibliography

Bibliography

- Adcroft, A. and J.-M. Campin, 2004: Rescaled height coordinates for accurate representation of free-surface flows in ocean circulation models. *Ocean Modelling*, **7**, 269–284, doi:10.1016/j.ocemod.2003.09.003.
- Haidvogel, D. and A. Beckmann, 1999: *Numerical Ocean Circulation Modeling*. Imperial College Press, London.
- Higdon, R. L., 2005: A two-level time-stepping method for layered ocean circulation models: further development and testing. *Journal of Computational Physics*, **206**, 463–504, doi:10.1016/j.jcp.2004.12.011.
- Ilicak, M., A. J. Adcroft, S. M. Griffies, and R. W. Hallberg, 2012: Spurious dianeutral mixing and the role of momentum closure. *Ocean Modelling*, **45**, 37–58, doi:10.1016/j.ocemod.2011.10.003.
- Jackett, D. R. and T. J. McDougall, 1995: Minimal Adjustment of Hydrographic Profiles to Achieve Static Stability. *Journal of Atmospheric and Oceanic Technology*, **12**, 381, doi:10.1175/1520-0426(1995)012j0381:MAOHPT;2.0.CO;2.
- Leith, C., 1996: Stochastic models of chaotic systems. *Physica D: Nonlinear Phenomena*, **98**, 481–491.
- Pacanowski, R. and S. Philander, 1981: Parameterization of vertical mixing in numerical models of tropical oceans. *J. Phys. Oceanography*, **11**, 1443–1451.
- Ringler, T., L. Ju, and M. Gunzburger, 2008: A multiresolution method for climate system modeling: application of spherical centroidal Voronoi tessellations. *Ocean Dynamics*, **58**, 475–498, doi:10.1007/s10236-008-0157-2.
- Ringler, T., M. Petersen, R. Higdon, D. Jacobsen, P. Jones, and M. Maltrud, 2013: A Multiresolution Approach to Global Ocean Modeling, in press.
- Ringler, T. D., D. Jacobsen, M. Gunzburger, L. Ju, M. Duda, and W. Skamarock, 2011: Exploring a Multiresolution Modeling Approach within the Shallow-Water Equations. *Monthly Weather Review*, **139**, 3348–3368, doi:10.1175/MWR-D-10-05049.1.
- Ringler, T. D., J. Thuburn, J. B. Klemp, and W. C. Skamarock, 2010: A unified approach to energy conservation and potential vorticity dynamics for arbitrarily-structured C-grids. *J. Comp. Physics*, **229**, 3065–3090, doi:10.1016/j.jcp.2009.12.007.
- Skamarock, W. and A. Gassmann, 2011: Conservative Transport Schemes for Spherical Geodesic Grids: High-Order Flux Operators for ODE-Based Time Integration. *Monthly Weather Review*, **139**, 2962–2975.

Smagorinsky, J., 1963: General circulation experiments with the primitive equations. *Monthly Weather Review*, **91**, 99–164.

Thuburn, J., T. D. Ringler, W. C. Skamarock, and J. B. Klemp, 2009: Numerical representation of geostrophic modes on arbitrarily structured C-grids. *Journal of Computational Physics*, **228**, 8321–8335, doi:10.1016/j.jcp.2009.08.006.

Vallis, G. K. and B.-L. Hua, 1988: Eddy viscosity of the anticipated potential vorticity method. *Journal of Atmospheric Sciences*, **45**, 617–627, doi:10.1175/1520-0469(1988)045;0617:EVOTAP;2.0.CO;2.

Zalesak, S., 1979: Fully multidimensional flux-corrected transport algorithms for fluids. *Journal of Computational Physics*, **31**, 335–362.

Part IV

Appendices

Appendix A

Namelist options

Embedded links point to information in chapter [7](#)

A.1 [time_management](#)

A.1.1 [config_do_restart](#)

Type:	logical
Units:	<i>unitless</i>
Default Value:	.false.
Possible Values:	.true. or .false.

Table A.1: config_do_restart: Determines if the initial conditions should be read from a restart file, or an input file.

A.1.2 [config_start_time](#)

Type:	character
Units:	<i>unitless</i>
Default Value:	0000-01-01_00:00:00
Possible Values:	'YYYY-MM-DD_HH:MM:SS' or 'file'

Table A.2: config_start_time: Timestamp describing the initial time of the simulation. If it is set to 'file', the initial time is read from restart_timestamp.

A.1.3 [config_stop_time](#)

Type:	character
Units:	<i>unitless</i>
Default Value:	none

Possible Values:	'YYYY-MM-DD HH:MM:SS' or 'none'
------------------	---------------------------------

Table A.3: config_stop_time: Timestamp describing the final time of the simulation. If it is set to 'none' the final time is determined from config_start_time and config_run_duration.

A.1.4 config_run_duration

Type:	character
Units:	<i>unitless</i>
Default Value:	0001.00:00:00
Possible Values:	'DDDD HH:MM:SS' or 'none'

Table A.4: config_run_duration: Timestamp describing the length of the simulation. If it is set to 'none' the duration is determined from config_start_time and config_stop_time. config_run_duration overrides inconsistent values of config_stop_time.

A.1.5 config_calendar_type

Type:	character
Units:	<i>unitless</i>
Default Value:	360day
Possible Values:	'gregorian', 'gregorian_noleap', or '360day'

Table A.5: config_calendar_type: Selection of the type of calendar that should be used in the simulation.

A.2 io

A.2.1 config_input_name

Type:	character
Units:	<i>unitless</i>
Default Value:	grid.nc
Possible Values:	path/to/grid.nc

Table A.6: config_input_name: The path to the input file for the simulation.

A.2.2 config_output_name

Type:	character
Units:	<i>unitless</i>
Default Value:	output.nc
Possible Values:	path/to/output.nc

Table A.7: config_output_name: The template path and name to the output file from the simulation. A time stamp is prepended to the extension of the file (.nc).

A.2.3 config_restart_name

Type:	character
Units:	<i>unitless</i>
Default Value:	restart.nc
Possible Values:	path/to/restart.nc

Table A.8: config_restart_name: The template path and name to the restart file for the simulation. A time stamp is prepended to the extension of the file (.nc) both for input and output.

A.2.4 config_restart_interval

Type:	character
Units:	<i>unitless</i>
Default Value:	0001_00:00:00
Possible Values:	'DDDD_HH:MM:SS'

Table A.9: config_restart_interval: Timestamp determining how often a restart file should be written.

A.2.5 config_output_interval

Type:	character
Units:	<i>unitless</i>
Default Value:	0001_00:00:00
Possible Values:	'DDDD_HH:MM:SS'

Table A.10: config_output_interval: Timestamp determining how often an output file should be written.

A.2.6 config_stats_interval

Type:	character
Units:	<i>unitless</i>
Default Value:	0000.01:00:00
Possible Values:	'DDDD_HH:MM:SS'

Table A.11: config_stats.interval: Timestamp determining how often a global statistics files should be written.

A.2.7 config_write_stats_on_startup

Type:	logical
Units:	<i>unitless</i>
Default Value:	.true.
Possible Values:	.true. or .false.

Table A.12: config_write_stats_on_startup: Logical flag determining if statistics files should be written prior to the first time step.

A.2.8 config_write_output_on_startup

Type:	logical
Units:	<i>unitless</i>
Default Value:	.true.
Possible Values:	.true. or .false.

Table A.13: config_write_output_on_startup: Logical flag determining if an output file should be written prior to the first time step.

A.2.9 config_frames_per_outfile

Type:	integer
Units:	<i>unitless</i>
Default Value:	1000
Possible Values:	Any positive integer value greater than 0

Table A.14: config_frames_per_outfile: Integer specifying how many time frames should be included in an output file. Once the maximum is reached, a new output file is created.

A.2.10 config_pio_num_iotasks

Type:	integer
Units:	<i>unitless</i>
Default Value:	0
Possible Values:	Any positive integer value greater than or equal to 0.

Table A.15: config_pio_num_iotasks: Integer specifying how many IO tasks should be used within the PIO library. A value of 0 causes all MPI tasks to also be IO tasks. IO tasks are required to write contiguous blocks of data to a file.

A.2.11 config_pio_stride

Type:	integer
Units:	<i>unitless</i>
Default Value:	1
Possible Values:	Any positive integer value greater than 0.

Table A.16: config_pio_stride: Integer specifying the stride of each IO task.

A.3 time_integration

A.3.1 config_dt

Type:	real
Units:	s
Default Value:	300.0
Possible Values:	Any positive real value, but limited by CFL condition.

Table A.17: config_dt: Length of model time-step.

A.3.2 config_time_integrator

Type:	character
Units:	<i>unitless</i>
Default Value:	split_explicit
Possible Values:	'split_explicit', 'RK4', 'unsplit_explicit'

Table A.18: config_time_integrator: Time integration method.

A.4 grid

A.4.1 config_num_halos

Type:	integer
Units:	<i>unitless</i>
Default Value:	3
Possible Values:	Any positive integer value.

Table A.19: config_num_halos: Determines the number of halo cells extending from a blocks owned cells (Called the 0-Halo). The default of 3 is the minimum that can be used with monotonic advection.

A.4.2 config_vert_coord_movement

Type:	character
Units:	<i>unitless</i>
Default Value:	uniform_stretching
Possible Values:	'uniform_stretching', 'fixed', 'user_specified', 'isopycnal'

Table A.20: config_vert_coord_movement: Determines the vertical coordinate movement type. 'uniform_stretching' distributes SSH perturbations through all vertical levels, 'fixed' places them all in the top level, 'user_specified' allows the input file to determine the distribution, and 'isopycnal' causes levels to be pure isopycnal.

A.4.3 config_alter_ICs_for_pbcs

Type:	character
Units:	<i>unitless</i>
Default Value:	zlevel_pbcs_off
Possible Values:	'zlevel_pbcs_on', 'zlevel_pbcs_off', 'off'

Table A.21: config_alter_ICs_for_pbc: Determines the method of alteration for partial bottom cells. 'zlevel_pbc_on' alters the initial conditions for partial bottom cells, 'zlevel_pbc_off' alters the initial conditions to have full cells everywhere, and 'off' does nothing to the initial conditions.

A.4.4 config_min_pbc_fraction

Type:	real
Units:	<i>unitless</i>
Default Value:	0.10
Possible Values:	Any real between 0 and 1.

Table A.22: config_min_pbc_fraction: Determines the minimum fraction of a cell altering the initial conditions can create.

A.4.5 config_check_ssh_consistency

Type:	logical
Units:	<i>unitless</i>
Default Value:	.true.
Possible Values:	.true. or .false.

Table A.23: config_check_ssh_consistency: Enables a check to determine if the SSH is consistent across relevant variables.

A.5 decomposition

A.5.1 config_block_decomp_file_prefix

Type:	character
Units:	<i>unitless</i>
Default Value:	graph.info.part.
Possible Values:	Any path/prefix to a block decomposition file.

Table A.24: config_block_decomp_file_prefix: Defines the prefix for the block decomposition file. Can include a path. The number of blocks is appended to the end of the prefix at run-time.

A.5.2 config_number_of_blocks

Type:	integer
Units:	<i>unitless</i>
Default Value:	0
Possible Values:	Any integer ≥ 0 .

Table A.25: config_number_of_blocks: Determines the number of blocks a simulation should be run with. If it is set to 0, the number of blocks is the same as the number of MPI tasks at run-time.

A.5.3 config_explicit_proc_decomp

Type:	logical
Units:	<i>unitless</i>
Default Value:	.false.
Possible Values:	.true. or .false.

Table A.26: config_explicit_proc_decomp: Determines if an explicit processor decomposition should be used. This is only useful if multiple blocks per processor are used.

A.5.4 config_proc_decomp_file_prefix

Type:	character
Units:	<i>unitless</i>
Default Value:	graph.info.part.
Possible Values:	Any path/prefix to a processor decomposition file.

Table A.27: config_proc_decomp_file_prefix: Defines the prefix for the processor decomposition file. This file is only read if config_explicit_proc_decomp is .true. The number of processors is appended to the end of the prefix at run-time.

A.6 hmix

A.6.1 config_hmix_ScaleWithMesh

Type:	logical
Units:	<i>unitless</i>
Default Value:	.false.

Possible Values:	.true. or .false.
------------------	-------------------

Table A.28: config_hmix_ScaleWithMesh: If false, del2 and del4 coefficients are constant throughout the mesh (equivalent to setting $\rho_m = 1$ throughout the mesh). If true, these coefficients scale as mesh density to the -3/4 power.

A.6.2 config_maxMeshDensity

Type:	real
Units:	<i>unitless</i>
Default Value:	-1.0
Possible Values:	Any positive real number. If set any negative real number, config_maxMeshDensity is computed during the initialization of each simulation.

Table A.29: config_maxMeshDensity: Global maximum of the mesh density

A.6.3 config_apvm_scale_factor

Type:	real
Units:	<i>unitless</i>
Default Value:	0.0
Possible Values:	Any non-negative number, typically between zero and one.

Table A.30: config_apvm_scale_factor: Anticipated potential vorticity (APV) method scale factor, c_{apv} . When zero, APV is off.

A.7 hmix_del2

A.7.1 config_use_mom_del2

Type:	logical
Units:	<i>unitless</i>
Default Value:	.true.
Possible Values:	.true. or .false.

Table A.31: config_use_mom_del2: If true, Laplacian horizontal mixing is used on the momentum equation.

A.7.2 config_use_tracer_del2

Type:	logical
Units:	<i>unitless</i>
Default Value:	.false.
Possible Values:	.true. or .false.

Table A.32: config_use_tracer_del2: If true, Laplacian horizontal mixing is used on the tracer equation.

A.7.3 config_mom_del2

Type:	real
Units:	$m^2 s^{-1}$
Default Value:	10.0
Possible Values:	any positive real

Table A.33: config_mom_del2: Horizontal viscosity, ν_h .

A.7.4 config_tracer_del2

Type:	real
Units:	$m^2 s^{-1}$
Default Value:	10.0
Possible Values:	any positive real

Table A.34: config_tracer_del2: Horizontal diffusion, κ_h .

A.8 hmix_del4

A.8.1 config_use_mom_del4

Type:	logical
Units:	<i>unitless</i>
Default Value:	.false.
Possible Values:	.true. or .false.

Table A.35: config_use_mom_del4: If true, biharmonic horizontal mixing is used on the momentum equation.

A.8.2 config_use_tracer_del4

Type:	logical
Units:	<i>unitless</i>
Default Value:	.false.
Possible Values:	.true. or .false.

Table A.36: config_use_tracer_del4: If true, biharmonic horizontal mixing is used on the tracer equation.

A.8.3 config_mom_del4

Type:	real
Units:	$m^4 s^{-1}$
Default Value:	5.0e13
Possible Values:	any positive real

Table A.37: config_mom_del4: Coefficient for horizontal biharmonic operator on momentum.

A.8.4 config_tracer_del4

Type:	real
Units:	$m^4 s^{-1}$
Default Value:	0.0
Possible Values:	any positive real

Table A.38: config_tracer_del4: Coefficient for horizontal biharmonic operator on tracers.

A.9 hmix_Leith

A.9.1 config_use_Leith_del2

Type:	logical
Units:	<i>unitless</i>
Default Value:	.false.
Possible Values:	.true. or .false.

Table A.39: config_use_Leith_del2: If true, the Leith enstrophy-cascade closure is turned on

A.9.2 config_Leith_parameter

Type:	real
Units:	<i>non-dimensional</i>
Default Value:	1.0
Possible Values:	any positive real

Table A.40: config_Leith_parameter: Non-dimensional Leith closure parameter

A.9.3 config_Leith_dx

Type:	real
Units:	m
Default Value:	15000.0
Possible Values:	any positive real

Table A.41: config_Leith_dx: Characteristic length scale, usually the smallest dx in the mesh

A.9.4 config_Leith_visc2_max

Type:	real
Units:	$m^2 s^{-1}$
Default Value:	2.5e3
Possible Values:	any positive real

Table A.42: config_Leith_visc2_max: Upper bound on the allowable value of Leith-computed viscosity

A.10 standard_GM

A.10.1 config_h_kappa

Type:	real
Units:	MISSING
Default Value:	0.0
Possible Values:	Any positive real value.

Table A.43: config_h_kappa: kappa parameter for Gent-McWilliams eddy parameterization

A.10.2 config_h_kappa_q

Type:	real
Units:	MISSING
Default Value:	0.0
Possible Values:	Any positive real value.

Table A.44: config_h_kappa_q: kappa-q parameter for Gent-McWilliams eddy parameterization

A.11 Rayleigh_damping

A.11.1 config_Rayleigh_friction

Type:	logical
Units:	<i>unitless</i>
Default Value:	.false.
Possible Values:	.true. or .false.

Table A.45: config_Rayleigh_friction: If true, Rayleigh friction is included in the momentum equation.

A.11.2 config_Rayleigh_damping_coeff

Type:	real
Units:	s^{-1}
Default Value:	0.0
Possible Values:	Any positive real value.

Table A.46: config_Rayleigh_damping_coeff: Inverse-time coefficient for the Rayleigh damping term, c_R .

A.12 `vmix`

A.12.1 `config_convective_visc`

Type:	real
Units:	$m^2 s^{-1}$
Default Value:	1.0
Possible Values:	Any positive real value.

Table A.47: `config_convective_visc`: Value of vertical viscosity to be used in unstable conditions (i.e. Richardson number less than zero)

A.12.2 `config_convective_diff`

Type:	real
Units:	$m^2 s^{-1}$
Default Value:	1.0
Possible Values:	Any positive real value.

Table A.48: `config_convective_diff`: Value of vertical diffusion to be used in unstable conditions (i.e. Richardson number less than zero)

A.13 `vmix_const`

A.13.1 `config_use_const_visc`

Type:	logical
Units:	<i>unitless</i>
Default Value:	.true.
Possible Values:	.true. or .false.

Table A.49: `config_use_const_visc`: If true, constant vertical viscosity is included in the momentum equation

A.13.2 `config_use_const_diff`

Type:	logical
Units:	<i>unitless</i>

Default Value:	.true.
Possible Values:	.true. or .false.

Table A.50: config_use_const_diff: If true, constant vertical diffusion is included in the tracer equation

A.13.3 config_vert_vis

Type:	real
Units:	$m^2 s^{-1}$
Default Value:	1.0e-4
Possible Values:	Any positive real value.

Table A.51: config_vert_vis: Vertical viscosity, applied uniformly throughout domain

A.13.4 config_vert_diff

Type:	real
Units:	$m^2 s^{-1}$
Default Value:	1.0e-4
Possible Values:	Any positive real value.

Table A.52: config_vert_diff: Vertical diffusion, applied uniformly throughout domain

A.14 vmix_rich

A.14.1 config_use_rich_vis

Type:	logical
Units:	<i>unitless</i>
Default Value:	.true.
Possible Values:	.true. or .false.

Table A.53: config_use_rich_vis: If true, Richardson-number based vertical viscosity is included in the momentum equation

A.14.2 config_use_rich_diff

Type:	logical
Units:	<i>unitless</i>
Default Value:	.true.
Possible Values:	.true. or .false.

Table A.54: config_use_rich_diff: If true, Richardson-number based vertical diffusion is included in the tracer equation

A.14.3 config_bkrd_vert_visc

Type:	real
Units:	$m^2 s^{-1}$
Default Value:	1.0e-4
Possible Values:	Any positive real value.

Table A.55: config_bkrd_vert_visc: Background vertical viscosity for Richardson-number based vertical mixing, ν_{bkrd}

A.14.4 config_bkrd_vert_diff

Type:	real
Units:	$m^2 s^{-1}$
Default Value:	1.0e-5
Possible Values:	Any positive real value.

Table A.56: config_bkrd_vert_diff: Background vertical diffusion for Richardson-number based vertical mixing, κ_{bkrd}

A.14.5 config_rich_mix

Type:	real
Units:	$m^2 s^{-1}$
Default Value:	0.005
Possible Values:	Any positive real value.

Table A.57: config_rich_mix: Coefficient for Richardson-number vertical mixing function, c_{Ri}

A.15 `vmix_tanh`

A.15.1 `config_use_tanh_visc`

Type:	logical
Units:	<i>unitless</i>
Default Value:	.false.
Possible Values:	.true. or .false.

Table A.58: `config_use_tanh_visc`: If true, tanh-based vertical viscosity is included in the momentum equation

A.15.2 `config_use_tanh_diff`

Type:	logical
Units:	<i>unitless</i>
Default Value:	.false.
Possible Values:	.true. or .false.

Table A.59: `config_use_tanh_diff`: If true, tanh-based vertical diffusion is included in the tracer equation

A.15.3 `config_max_visc_tanh`

Type:	real
Units:	$m^2 s^{-1}$
Default Value:	2.5e-1
Possible Values:	Any positive real value.

Table A.60: `config_max_visc_tanh`: maximum viscosity value for tanh-fit function

A.15.4 `config_min_visc_tanh`

Type:	real
Units:	$m^2 s^{-1}$
Default Value:	1.0e-4
Possible Values:	Any positive real value.

Table A.61: `config_min_visc_tanh`: minimum viscosity value for tanh-fit function

A.15.5 config_max_diff_tanh

Type:	real
Units:	$m^2 s^{-1}$
Default Value:	2.5e-2
Possible Values:	Any positive real value.

Table A.62: config_max_diff_tanh: maximum diffusion value for tanh-fit function

A.15.6 config_min_diff_tanh

Type:	real
Units:	$m^2 s^{-1}$
Default Value:	1.0e-5
Possible Values:	Any positive real value.

Table A.63: config_min_diff_tanh: minimum diffusion value for tanh-fit function

A.15.7 config_zMid_tanh

Type:	real
Units:	m
Default Value:	-100
Possible Values:	Any real value, typically negative.

Table A.64: config_zMid_tanh: z-coodinate location of center of tanh function

A.15.8 config_zWidth_tanh

Type:	real
Units:	m
Default Value:	100
Possible Values:	Any positive real value.

Table A.65: config_zWidth_tanh: vertical width parameter for tanh function

A.16 forcing

A.16.1 config_use_monthly_forcing

Type:	logical
Units:	<i>unitless</i>
Default Value:	.false.
Possible Values:	.true. or .false.

Table A.66: config_use_monthly_forcing: Controls time frequency of forcing. If false, a constant forcing is used, provided by the input fields normalVelocityForcing, temperatureRestore, and salinityRestore. If true, forcing is interpolated between monthly fields given by windStressMonthly, temperatureRestoreMonthly, and salinityRestoreMonthly.

A.16.2 config_restoreTS

Type:	logical
Units:	<i>unitless</i>
Default Value:	.false.
Possible Values:	.true. or .false.

Table A.67: config_restoreTS: If true, the restoring term is activated in the tracer equation for temperature and salinity.

A.16.3 config_restoreT_timescale

Type:	real
Units:	<i>days</i>
Default Value:	90.0
Possible Values:	any positive real value, but typically between 30 and 90 days.

Table A.68: config_restoreT_timescale: Restoring timescale for temperature, τ_r .

A.16.4 config_restoreS_timescale

Type:	real
Units:	<i>days</i>
Default Value:	90.0
Possible Values:	any positive real value, but typically between 30 and 90 days.

Table A.69: config_restoreS_timescale: Restoring timescale for salinity, τ_r .

A.17 advection

A.17.1 config_vert_tracer_adv

Type:	character
Units:	<i>unitless</i>
Default Value:	stencil
Possible Values:	'spline' and 'stencil'

Table A.70: config_vert_tracer_adv: Method for interpolating tracer values from layer centers to layer edges

A.17.2 config_vert_tracer_adv_order

Type:	integer
Units:	<i>unitless</i>
Default Value:	3
Possible Values:	2, 3 and 4

Table A.71: config_vert_tracer_adv_order: Order of polynomial used for tracer reconstruction at layer edges

A.17.3 config_horiz_tracer_adv_order

Type:	integer
Units:	<i>unitless</i>
Default Value:	3
Possible Values:	2, 3 and 4

Table A.72: config_horiz_tracer_adv_order: Order of polynomial used for tracer reconstruction at cell edges

A.17.4 config_coef_3rd_order

Type:	real
Units:	<i>non-dimensional</i>
Default Value:	0.25
Possible Values:	any real between 0 and 1

Table A.73: config_coef_3rd_order: Reconstruction of 3rd-order reconstruction to blend with 4th-order reconstruction

A.17.5 config_monotonic

Type:	logical
Units:	<i>unitless</i>
Default Value:	.true.
Possible Values:	.true. and .false.

Table A.74: config_monotonic: If .true. then fluxes are limited to produce a monotonic advection scheme

A.18 bottom_drag

A.18.1 config_bottom_drag_coeff

Type:	real
Units:	<i>unitless</i>
Default Value:	1.0e-2
Possible Values:	any positive real, typically 1.0e-3

Table A.75: config_bottom_drag_coeff: Dimensionless bottom drag coefficient, c_{drag} .

A.19 pressure_gradient

A.19.1 config_pressure_gradient_type

Type:	character
Units:	<i>unitless</i>
Default Value:	pressure_and_zmid
Possible Values:	'pressure_and_zmid' or 'MontgomeryPotential'

Table A.76: config_pressure_gradient_type: Form of pressure gradient terms in momentum equation. For most applications, the gradient of pressure and layer mid-depth are appropriate. For isopycnal coordinates, one may use the gradient of the Montgomery potential.

A.19.2 config_density0

Type:	real
Units:	$kg\ m^{-3}$
Default Value:	1014.65
Possible Values:	any positive real, but typically 1000-1035

Table A.77: config_density0: Density used as a coefficient of the pressure gradient terms, ρ_0 . This is a constant due to the Boussinesq approximation.

A.20 eos

A.20.1 config_eos_type

Type:	character
Units:	<i>unitless</i>
Default Value:	linear
Possible Values:	Jackett McDougall EOS = 'jm' and Linear EOS = 'linear'

Table A.78: config_eos_type: Character string to choose EOS formulation

A.21 eos_linear

A.21.1 config_eos_linear_alpha

Type:	real
Units:	$kg\ m^{-3}\ C^{-1}$
Default Value:	2.55e-1
Possible Values:	any positive real

Table A.79: config_eos_linear_alpha: Linear thermal expansion coefficient

A.21.2 config_eos_linear_beta

Type:	real
Units:	$kg\ m^{-3}\ PSU^{-1}$
Default Value:	7.64e-1
Possible Values:	any positive real

Table A.80: config_eos_linear_beta: Linear haline contraction coefficient

A.21.3 config_eos_linear_Tref

Type:	real
Units:	C
Default Value:	19.0
Possible Values:	any real

Table A.81: config_eos_linear_Tref: Reference temperature

A.21.4 config_eos_linear_Sref

Type:	real
Units:	PSU
Default Value:	35.0
Possible Values:	any real

Table A.82: config_eos_linear_Sref: Reference salinity

A.21.5 config_eos_linear_densityref

Type:	real
Units:	$kg\ m^{-3}$
Default Value:	1025.022
Possible Values:	any positive real

Table A.83: config_eos_linear_densityref: Reference density, i.e. density when T=Tref and S=Sref

A.22 `split_explicit_ts`

A.22.1 `config_n_ts_iter`

Type:	integer
Units:	<i>unitless</i>
Default Value:	2
Possible Values:	any positive integer, but typically 1, 2, or 3

Table A.84: `config_n_ts_iter`: number of large iterations over stages 1-3

A.22.2 `config_n_bcl_iter_beg`

Type:	integer
Units:	<i>unitless</i>
Default Value:	1
Possible Values:	any positive integer, but typically 1, 2, or 3

Table A.85: `config_n_bcl_iter_beg`: number of iterations of stage 1 (baroclinic solve) on the first split-explicit iteration

A.22.3 `config_n_bcl_iter_mid`

Type:	integer
Units:	<i>unitless</i>
Default Value:	2
Possible Values:	any positive integer, but typically 1, 2, or 3

Table A.86: `config_n_bcl_iter_mid`: number of iterations of stage 1 (baroclinic solve) on any split-explicit iterations between first and last

A.22.4 `config_n_bcl_iter_end`

Type:	integer
Units:	<i>unitless</i>
Default Value:	2
Possible Values:	any positive integer, but typically 1, 2, or 3

Table A.87: `config_n_bcl_iter_end`: number of iterations of stage 1 (baroclinic solve) on the last split-explicit iteration

A.22.5 config_n_btr_subcycles

Type:	integer
Units:	<i>unitless</i>
Default Value:	20
Possible Values:	any positive integer, typically between 10 and 100

Table A.88: config_n_btr_subcycles: number of barotropic subcycles in stage 2

A.22.6 config_n_btr_cor_iter

Type:	integer
Units:	<i>unitless</i>
Default Value:	2
Possible Values:	any positive integer, but typically 1, 2, or 3

Table A.89: config_n_btr_cor_iter: number of iterations of the velocity corrector step in stage 2

A.22.7 config_vel_correction

Type:	logical
Units:	<i>unitless</i>
Default Value:	.true.
Possible Values:	.true. or .false.

Table A.90: config_vel_correction: If true, the velocity correction term is included in the horizontal advection of thickness and tracers

A.22.8 config_btr_subcycle_loop_factor

Type:	integer
Units:	<i>unitless</i>
Default Value:	2
Possible Values:	Any positive integer, but typically 1 or 2

Table A.91: config_btr_subcycle_loop_factor: Barotropic subcycles proceed from t to $t + n\Delta t$, where n is this configuration option.

A.22.9 config_btr_gam1_velWt1

Type:	real
Units:	<i>unitless</i>
Default Value:	0.5
Possible Values:	between 0 and 1

Table A.92: config_btr_gam1_velWt1: Weighting of velocity in the SSH predictor step in stage 2. When zero, previous subcycle time is used; when one, new subcycle time is used.

A.22.10 config_btr_gam2_SSHWt1

Type:	real
Units:	<i>unitless</i>
Default Value:	1.0
Possible Values:	between 0 and 1

Table A.93: config_btr_gam2_SSHWt1: Weighting of SSH in the velocity corrector step in stage 2. When zero, previous subcycle time is used; when one, new subcycle time is used.

A.22.11 config_btr_gam3_velWt2

Type:	real
Units:	<i>unitless</i>
Default Value:	1.0
Possible Values:	between 0 and 1

Table A.94: config_btr_gam3_velWt2: Weighting of velocity in the SSH corrector step in stage 2. When zero, previous subcycle time is used; when one, new subcycle time is used.

A.22.12 config_btr_solve_SSH2

Type:	logical
Units:	<i>unitless</i>
Default Value:	.false.

Possible Values:	.true. or .false.
------------------	-------------------

Table A.95: config_btr_solve_SSH2: If true, execute the SSH corrector step in stage 2

A.23 debug

A.23.1 config_check_zlevel_consistency

Type:	logical
Units:	<i>unitless</i>
Default Value:	.false.
Possible Values:	.true. or .false.

Table A.96: config_check_zlevel_consistency: Enables a run-time check for consistency for a zlevel grid. Ensures relevant variables correctly define the bottom of the ocean.

A.23.2 config_filter_btr_mode

Type:	logical
Units:	<i>unitless</i>
Default Value:	.false.
Possible Values:	.true. or .false.

Table A.97: config_filter_btr_mode: Enables filtering of the barotropic mode.

A.23.3 config_prescribe_velocity

Type:	logical
Units:	<i>unitless</i>
Default Value:	.false.
Possible Values:	.true. or .false.

Table A.98: config_prescribe_velocity: Enables a prescribed velocity field. This velocity field is read on input, and remains constant through a simulation.

A.23.4 config_prescribe_thickness

Type:	logical
Units:	<i>unitless</i>
Default Value:	.false.
Possible Values:	.true. or .false.

Table A.99: config_prescribe_thickness: Enables a prescribed thickness field. This thickness field is read on input, and remains constant through a simulation.

A.23.5 config_include_KE_vertex

Type:	logical
Units:	<i>unitless</i>
Default Value:	.false.
Possible Values:	.true. or .false.

Table A.100: config_include_KE_vertex: If true, the kinetic energy in each cell is computed by blending cell-based and vertex-based values of kinetic energy.

A.23.6 config_check_tracer_monotonicity

Type:	logical
Units:	<i>unitless</i>
Default Value:	.false.
Possible Values:	.true. or .false.

Table A.101: config_check_tracer_monotonicity: Enables a change on tracer monotonicity at the end of the monotonic advection routine. Only used if config_monotonic is set to .true.

A.23.7 config_disable_thick_all_tend

Type:	logical
Units:	<i>unitless</i>
Default Value:	.false.
Possible Values:	.true. or .false.

Table A.102: config_disable_thick_all_tend: Disables all tendencies on the thickness field.

A.23.8 config_disable_thick_hadv

Type:	logical
Units:	<i>unitless</i>
Default Value:	.false.
Possible Values:	.true. or .false.

Table A.103: config_disable_thick_hadv: Disable tendencies on the thickness field from horizontal advection.

A.23.9 config_disable_thick_vadv

Type:	logical
Units:	<i>unitless</i>
Default Value:	.false.
Possible Values:	.true. or .false.

Table A.104: config_disable_thick_vadv: Disables tendencies on the thickness field from vertical advection.

A.23.10 config_disable_vel_all_tend

Type:	logical
Units:	<i>unitless</i>
Default Value:	.false.
Possible Values:	.true. or .false.

Table A.105: config_disable_vel_all_tend: Disables all tendencies on the velocity field.

A.23.11 config_disable_vel_coriolis

Type:	logical
Units:	<i>unitless</i>
Default Value:	.false.
Possible Values:	.true. or .false.

Table A.106: config_disable_vel_coriolis: Diables tendencies on the velocity field from the Coriolis force.

A.23.12 config_disable_vel_pgrad

Type:	logical
Units:	<i>unitless</i>
Default Value:	.false.
Possible Values:	.true. or .false.

Table A.107: config_disable_vel_pgrad: Disables tendencies on the velocity field from the horizontal pressure gradient.

A.23.13 config_disable_vel_hmix

Type:	logical
Units:	<i>unitless</i>
Default Value:	.false.
Possible Values:	.true. or .false.

Table A.108: config_disable_vel_hmix: Disables tendencies on the velocity field from horizontal mixing.

A.23.14 config_disable_vel_windstress

Type:	logical
Units:	<i>unitless</i>
Default Value:	.false.
Possible Values:	.true. or .false.

Table A.109: config_disable_vel_windstress: Disables tendencies on the velocity field from horizontal wind stress.

A.23.15 config_disable_vel_vmix

Type:	logical
Units:	<i>unitless</i>
Default Value:	.false.
Possible Values:	.true. or .false.

Table A.110: config_disable_vel_vmix: Disables tendencies on the velocity field from vertical mixing.

A.23.16 config_disable_vel_vadv

Type:	logical
Units:	<i>unitless</i>
Default Value:	.false.
Possible Values:	.true. or .false.

Table A.111: config_disable_vel_vadv: Disables tendencies on the velocity field from vertical advection.

A.23.17 config_disable_tr_all_tend

Type:	logical
Units:	<i>unitless</i>
Default Value:	.false.
Possible Values:	.true. or .false.

Table A.112: config_disable_tr_all_tend: Disables all tendencies on tracer fields.

A.23.18 config_disable_tr_adv

Type:	logical
Units:	<i>unitless</i>
Default Value:	.false.
Possible Values:	.true. or .false.

Table A.113: config_disable_tr_adv: Disables tendencies on tracer fields from advection, both horizontal and vertical.

A.23.19 config_disable_tr_hmix

Type:	logical
Units:	<i>unitless</i>
Default Value:	.false.
Possible Values:	.true. or .false.

Table A.114: config_disable_tr_hmix: Disables tendencies on tracer fields from horizontal mixing.

A.23.20 config_disable_tr_vmix

Type:	logical
Units:	<i>unitless</i>
Default Value:	.false.
Possible Values:	.true. or .false.

Table A.115: config_disable_tr_vmix: Disables tendencies on tracer fields from vertical mixing.

Appendix B

Variable definitions

Embedded links point to information in chapter [8](#)

B.1 [state](#)

B.1.1 [temperature](#)

Type:	real
Units:	<i>degrees Celsius</i>
Dimension:	nVertLevels nCells Time
Persistence:	persistent
Default Streams:	Input Restart Output
Index in tracers Array:	domain % blocklist % state % index_temperature
Location in code:	domain % blocklist % state % time_levs(:) % state % tracers
Array Group:	dynamics

Table B.1: temperature: potential temperature

B.1.2 [salinity](#)

Type:	real
Units:	<i>grams salt per kilogram seawater</i>
Dimension:	nVertLevels nCells Time
Persistence:	persistent
Default Streams:	Input Restart Output
Index in tracers Array:	domain % blocklist % state % index_salinity
Location in code:	domain % blocklist % state % time_levs(:) % state % tracers
Array Group:	dynamics

Table B.2: salinity: salinity

B.1.3 `xtime`

Type:	text
Units:	<i>unitless</i>
Dimension:	Time
Persistence:	persistent
Default Streams:	Restart Output
Location in code:	domain % blocklist % state % time_levs(:) % state % xtime

Table B.3: `xtime`: model time, with format 'YYYY-MM-DD HH:MM:SS'

B.1.4 `normalVelocity`

Type:	real
Units:	$m\ s^{-1}$
Dimension:	nVertLevels nEdges Time
Persistence:	persistent
Default Streams:	Input Restart
Location in code:	domain % blocklist % state % time_levs(:) % state % normalVelocity

Table B.4: `normalVelocity`: horizontal velocity, normal component to an edge

B.1.5 `layerThickness`

Type:	real
Units:	m
Dimension:	nVertLevels nCells Time
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % state % time_levs(:) % state % layerThickness

Table B.5: `layerThickness`: layer thickness

B.1.6 `density`

Type:	real
Units:	$kg\ m^{-3}$
Dimension:	nVertLevels nCells Time

Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % state % time_levs(:) % state % density

Table B.6: density: density

B.1.7 [normalBarotropicVelocity](#)

Type:	real
Units:	$m s^{-1}$
Dimension:	nEdges Time
Persistence:	persistent
Default Streams:	Restart
Location in code:	domain % blocklist % state % time_levs(:) % state % normalBarotropicVelocity

Table B.7: [normalBarotropicVelocity](#): barotropic velocity, used in split-explicit time-stepping

B.1.8 [ssh](#)

Type:	real
Units:	m
Dimension:	nCells Time
Persistence:	persistent
Default Streams:	Output
Location in code:	domain % blocklist % state % time_levs(:) % state % ssh

Table B.8: [ssh](#): sea surface height

B.1.9 [normalBarotropicVelocitySubcycle](#)

Type:	real
Units:	$m s^{-1}$
Dimension:	nEdges Time
Persistence:	persistent
Default Streams:	None
Location in code:	domain % blocklist % state % time_levs(:) % state % normalBarotropicVelocitySubcycle

Table B.9: `normalBarotropicVelocitySubcycle`: barotropic velocity, used in subcycling in stage 2 of split-explicit time-stepping

B.1.10 `sshSubcycle`

Type:	real
Units:	m
Dimension:	nCells Time
Persistence:	persistent
Default Streams:	None
Location in code:	domain % blocklist % state % time_levs(:) % state % ssh-Subcycle

Table B.10: `sshSubcycle`: sea surface height, used in subcycling in stage 2 of split-explicit time-stepping

B.1.11 `barotropicThicknessFlux`

Type:	real
Units:	$m^2 s^{-1}$
Dimension:	nEdges Time
Persistence:	persistent
Default Streams:	None
Location in code:	domain % blocklist % state % time_levs(:) % state % barotropicThicknessFlux

Table B.11: `barotropicThicknessFlux`: Barotropic thickness flux at each edge, used to advance sea surface height in each subcycle of stage 2 of the split-explicit algorithm.

B.1.12 `barotropicForcing`

Type:	real
Units:	$m s^{-2}$
Dimension:	nEdges Time
Persistence:	persistent
Default Streams:	None
Location in code:	domain % blocklist % state % time_levs(:) % state % barotropicForcing

Table B.12: `barotropicForcing`: Barotropic tendency computed from the baroclinic equations in stage 1 of the split-explicit algorithm.

B.1.13 `normalBaroclinicVelocity`

Type:	real
Units:	$m s^{-1}$
Dimension:	nVertLevels nEdges Time
Persistence:	persistent
Default Streams:	None
Location in code:	domain % blocklist % state % time_levs(:) % state % normalBaroclinicVelocity

Table B.13: `normalBaroclinicVelocity`: baroclinic velocity, used in split-explicit time-stepping

B.1.14 `zMid`

Type:	real
Units:	m
Dimension:	nVertLevels nCells Time
Persistence:	persistent
Default Streams:	None
Location in code:	domain % blocklist % state % time_levs(:) % state % zMid

Table B.14: `zMid`: z-coordinate of the mid-depth of the layer

B.1.15 `tangentialVelocity`

Type:	real
Units:	$m s^{-1}$
Dimension:	nVertLevels nEdges Time
Persistence:	persistent
Default Streams:	None
Location in code:	domain % blocklist % state % time_levs(:) % state % tangentialVelocity

Table B.15: `tangentialVelocity`: horizontal velocity, tangential to an edge

B.1.16 **uTransport**

Type:	real
Units:	$m s^{-1}$
Dimension:	nVertLevels nEdges Time
Persistence:	persistent
Default Streams:	None
Location in code:	domain % blocklist % state % time_levs(:) % state % uTransport

Table B.16: uTransport: horizontal velocity used to transport mass and tracers

B.1.17 **uBolusGM**

Type:	real
Units:	$m s^{-1}$
Dimension:	nVertLevels nEdges Time
Persistence:	persistent
Default Streams:	None
Location in code:	domain % blocklist % state % time_levs(:) % state % uBolusGM

Table B.17: uBolusGM: Bolus velocity in Gent-McWilliams eddy parameterization

B.1.18 **uBolusGMX**

Type:	real
Units:	$m s^{-1}$
Dimension:	nVertLevels nEdges Time
Persistence:	persistent
Default Streams:	None
Location in code:	domain % blocklist % state % time_levs(:) % state % uBolusGMX

Table B.18: uBolusGMX: Bolus velocity in Gent-McWilliams eddy parameterization, x-direction

B.1.19 **uBolusGMY**

Type:	real
Units:	$m s^{-1}$
Dimension:	nVertLevels nEdges Time
Persistence:	persistent
Default Streams:	None
Location in code:	domain % blocklist % state % time_levs(:) % state % uBolusGMY

Table B.19: uBolusGMY: Bolus velocity in Gent-McWilliams eddy parameterization, y-direction

B.1.20 **uBolusGMZ**

Type:	real
Units:	$m s^{-1}$
Dimension:	nVertLevels nEdges Time
Persistence:	persistent
Default Streams:	None
Location in code:	domain % blocklist % state % time_levs(:) % state % uBolusGMZ

Table B.20: uBolusGMZ: Bolus velocity in Gent-McWilliams eddy parameterization, z-direction

B.1.21 **uBolusGMZonal**

Type:	real
Units:	$m s^{-1}$
Dimension:	nVertLevels nEdges Time
Persistence:	persistent
Default Streams:	None
Location in code:	domain % blocklist % state % time_levs(:) % state % uBolusGMZonal

Table B.21: uBolusGMZonal: Bolus velocity in Gent-McWilliams eddy parameterization, zonal-direction

B.1.22 **uBolusGMMeridional**

Type:	real
-------	------

Units:	$m s^{-1}$
Dimension:	nVertLevels nEdges Time
Persistence:	persistent
Default Streams:	None
Location in code:	domain % blocklist % state % time_levs(:) % state % uBolusGMMeridional

Table B.22: uBolusGMMeridional: Bolus velocity in Gent-McWilliams eddy parameterization, meridional-direction

B.1.23 hEddyFlux

Type:	real
Units:	MISSING
Dimension:	nVertLevels nEdges Time
Persistence:	persistent
Default Streams:	None
Location in code:	domain % blocklist % state % time_levs(:) % state % hEddyFlux

Table B.23: hEddyFlux: Eddy flux in Gent-McWilliams eddy parameterization

B.1.24 h_kappa

Type:	real
Units:	MISSING
Dimension:	nVertLevels nEdges Time
Persistence:	persistent
Default Streams:	None
Location in code:	domain % blocklist % state % time_levs(:) % state % h_kappa

Table B.24: h_kappa: kappa parameter for Gent-McWilliams eddy parameterization

B.1.25 h_kappa_q

Type:	real
Units:	MISSING
Dimension:	nVertLevels nEdges Time
Persistence:	persistent

Default Streams:	None
Location in code:	domain % blocklist % state % time_levs(:) % state % h.kappa_q

Table B.25: h.kappa_q: kappa_q parameter for Gent-McWilliams eddy parameterization

B.1.26 [divergence](#)

Type:	real
Units:	s^{-1}
Dimension:	nVertLevels nCells Time
Persistence:	persistent
Default Streams:	Output
Location in code:	domain % blocklist % state % time_levs(:) % state % divergence

Table B.26: divergence: divergence of horizontal velocity

B.1.27 [relativeVorticity](#)

Type:	real
Units:	s^{-1}
Dimension:	nVertLevels nVertices Time
Persistence:	persistent
Default Streams:	
Location in code:	domain % blocklist % state % time_levs(:) % state % relativeVorticity

Table B.27: relativeVorticity: curl of horizontal velocity, defined at vertices

B.1.28 [relativeVorticityCell](#)

Type:	real
Units:	s^{-1}
Dimension:	nVertLevels nCells Time
Persistence:	persistent
Default Streams:	Output
Location in code:	domain % blocklist % state % time_levs(:) % state % relativeVorticityCell

Table B.28: relativeVorticityCell: curl of horizontal velocity, averaged from vertices to cell centers

B.1.29 normalizedRelativeVorticityEdge

Type:	real
Units:	s^{-1}
Dimension:	nVertLevels nEdges Time
Persistence:	persistent
Default Streams:	None
Location in code:	domain % blocklist % state % time_levs(:) % state % normalizedRelativeVorticityEdge

Table B.29: normalizedRelativeVorticityEdge: curl of horizontal velocity divided by layer thickness, averaged from vertices to edges

B.1.30 normalizedPlanetaryVorticityEdge

Type:	real
Units:	s^{-1}
Dimension:	nVertLevels nEdges Time
Persistence:	persistent
Default Streams:	None
Location in code:	domain % blocklist % state % time_levs(:) % state % normalizedPlanetaryVorticityEdge

Table B.30: normalizedPlanetaryVorticityEdge: earth's rotational rate (Coriolis parameter, f) divided by layer thickness, averaged from vertices to edges

B.1.31 normalizedRelativeVorticityCell

Type:	real
Units:	s^{-1}
Dimension:	nVertLevels nCells Time
Persistence:	persistent
Default Streams:	None
Location in code:	domain % blocklist % state % time_levs(:) % state % normalizedRelativeVorticityCell

Table B.31: normalizedRelativeVorticityCell: curl of horizontal velocity divided by layer thickness, averaged from vertices to cell centers

B.1.32 **layerThicknessEdge**

Type:	real
Units:	m
Dimension:	nVertLevels nEdges Time
Persistence:	persistent
Default Streams:	None
Location in code:	domain % blocklist % state % time_levs(:) % state % layerThicknessEdge

Table B.32: layerThicknessEdge: layer thickness averaged from cell center to edges

B.1.33 **layerThicknessVertex**

Type:	real
Units:	m
Dimension:	nVertLevels nVertices Time
Persistence:	persistent
Default Streams:	None
Location in code:	domain % blocklist % state % time_levs(:) % state % layerThicknessVertex

Table B.33: layerThicknessVertex: layer thickness averaged from cell center to vertices

B.1.34 **kineticEnergyCell**

Type:	real
Units:	$m^2 s^{-2}$
Dimension:	nVertLevels nCells Time
Persistence:	persistent
Default Streams:	Output
Location in code:	domain % blocklist % state % time_levs(:) % state % kineticEnergyCell

Table B.34: kineticEnergyCell: kinetic energy of horizontal velocity on cells

B.1.35 **normalVelocityX**

Type:	real
Units:	$m s^{-1}$
Dimension:	nVertLevels nCells Time
Persistence:	persistent
Default Streams:	None
Location in code:	domain % blocklist % state % time_levs(:) % state % normalVelocityX

Table B.35: normalVelocityX: component of horizontal velocity in the x-direction (cartesian)

B.1.36 **normalVelocityY**

Type:	real
Units:	$m s^{-1}$
Dimension:	nVertLevels nCells Time
Persistence:	persistent
Default Streams:	None
Location in code:	domain % blocklist % state % time_levs(:) % state % normalVelocityY

Table B.36: normalVelocityY: component of horizontal velocity in the y-direction (cartesian)

B.1.37 **normalVelocityZ**

Type:	real
Units:	$m s^{-1}$
Dimension:	nVertLevels nCells Time
Persistence:	persistent
Default Streams:	None
Location in code:	domain % blocklist % state % time_levs(:) % state % normalVelocityZ

Table B.37: normalVelocityZ: component of horizontal velocity in the z-direction (cartesian)

B.1.38 `normalVelocityZonal`

Type:	real
Units:	$m s^{-1}$
Dimension:	nVertLevels nCells Time
Persistence:	persistent
Default Streams:	Output
Location in code:	domain % blocklist % state % time_levs(:) % state % normalVelocityZonal

Table B.38: `normalVelocityZonal`: component of horizontal velocity in the eastward direction

B.1.39 `normalVelocityMeridional`

Type:	real
Units:	$m s^{-1}$
Dimension:	nVertLevels nCells Time
Persistence:	persistent
Default Streams:	Output
Location in code:	domain % blocklist % state % time_levs(:) % state % normalVelocityMeridional

Table B.39: `normalVelocityMeridional`: component of horizontal velocity in the northward

B.1.40 `montgomeryPotential`

Type:	real
Units:	$m^2 s^{-2}$
Dimension:	nVertLevels nCells Time
Persistence:	persistent
Default Streams:	None
Location in code:	domain % blocklist % state % time_levs(:) % state % montgomeryPotential

Table B.40: `montgomeryPotential`: Montgomery potential, may be used as the pressure for isopycnal coordinates.

B.1.41 `pressure`

Type:	real
Units:	$N m^{-2}$
Dimension:	nVertLevels nCells Time
Persistence:	persistent
Default Streams:	Output
Location in code:	domain % blocklist % state % time_levs(:) % state % pressure

Table B.41: pressure: pressure used in the momentum equation

B.1.42 **vertTransportVelocityTop**

Type:	real
Units:	$m s^{-1}$
Dimension:	nVertLevelsP1 nCells Time
Persistence:	persistent
Default Streams:	Output
Location in code:	domain % blocklist % state % time_levs(:) % state % vertTransportVelocityTop

Table B.42: vertTransportVelocityTop: vertical transport through the layer interface at the top of the cell

B.1.43 **vertVelocityTop**

Type:	real
Units:	$m s^{-1}$
Dimension:	nVertLevelsP1 nCells Time
Persistence:	persistent
Default Streams:	Output
Location in code:	domain % blocklist % state % time_levs(:) % state % vertVelocityTop

Table B.43: vertVelocityTop: vertical velocity defined at center (horizontally) and top (vertically) of cell

B.1.44 **displacedDensity**

Type:	real
Units:	$kg m^{-3}$
Dimension:	nVertLevels nCells Time

Persistence:	persistent
Default Streams:	Output
Location in code:	domain % blocklist % state % time_levs(:) % state % displacedDensity

Table B.44: displacedDensity: potential density displaced to the mid-depth of top layer

B.1.45 BruntVaisalaFreqTop

Type:	real
Units:	s^{-2}
Dimension:	nVertLevels nCells Time
Persistence:	persistent
Default Streams:	Output
Location in code:	domain % blocklist % state % time_levs(:) % state % BruntVaisalaFreqTop

Table B.45: BruntVaisalaFreqTop: Brunt Vaisala frequency defined at the center (horizontally) and top (vertically) of cell

B.1.46 viscosity

Type:	real
Units:	$m^2 s^{-1}$
Dimension:	nVertLevels nEdges Time
Persistence:	persistent
Default Streams:	Output
Location in code:	domain % blocklist % state % time_levs(:) % state % viscosity

Table B.46: viscosity: horizontal viscosity

B.1.47 circulation

Type:	real
Units:	$m^2 s^{-1}$
Dimension:	nVertLevels nVertices Time
Persistence:	persistent
Default Streams:	None
Location in code:	domain % blocklist % state % time_levs(:) % state % circulation

Table B.47: circulation: area-integrated vorticity

B.1.48 **areaCellGlobal**

Type:	real
Units:	m^2
Dimension:	Time
Persistence:	persistent
Default Streams:	Output
Location in code:	domain % blocklist % state % time_levs(:) % state % area-CellGlobal

Table B.48: areaCellGlobal: sum of the areaCell variable over the full domain, used to normalize global statistics

B.1.49 **areaEdgeGlobal**

Type:	real
Units:	m^2
Dimension:	Time
Persistence:	persistent
Default Streams:	Output
Location in code:	domain % blocklist % state % time_levs(:) % state % areaEdgeGlobal

Table B.49: areaEdgeGlobal: sum of the areaEdge variable over the full domain, used to normalize global statistics

B.1.50 **areaTriangleGlobal**

Type:	real
Units:	m^2
Dimension:	Time
Persistence:	persistent
Default Streams:	Output
Location in code:	domain % blocklist % state % time_levs(:) % state % area-TriangleGlobal

Table B.50: areaTriangleGlobal: sum of the areaTriangle variable over the full domain, used to normalize global statistics

B.1.51 **volumeCellGlobal**

Type:	real
Units:	m^3
Dimension:	Time
Persistence:	persistent
Default Streams:	Output
Location in code:	domain % blocklist % state % time_levs(:) % state % volumeCellGlobal

Table B.51: volumeCellGlobal: sum of the volumeCell variable over the full domain, used to normalize global statistics

B.1.52 **volumeEdgeGlobal**

Type:	real
Units:	m^3
Dimension:	Time
Persistence:	persistent
Default Streams:	Output
Location in code:	domain % blocklist % state % time_levs(:) % state % volumeEdgeGlobal

Table B.52: volumeEdgeGlobal: sum of the volumeEdge variable over the full domain, used to normalize global statistics

B.1.53 **CFLNumberGlobal**

Type:	real
Units:	<i>unitless</i>
Dimension:	Time
Persistence:	persistent
Default Streams:	Output
Location in code:	domain % blocklist % state % time_levs(:) % state % CFLNumberGlobal

Table B.53: CFLNumberGlobal: maximum CFL number over the full domain

B.1.54 nAverage

Type:	real
Units:	<i>unitless</i>
Dimension:	Time
Persistence:	persistent
Default Streams:	Output
Location in code:	domain % blocklist % state % time_levs(:) % state % nAverage

Table B.54: nAverage: number of timesteps in time-averaged variables

B.1.55 avgSsh

Type:	real
Units:	m
Dimension:	nCells Time
Persistence:	persistent
Default Streams:	Output
Location in code:	domain % blocklist % state % time_levs(:) % state % avgSsh

Table B.55: avgSsh: time-averaged sea surface height

B.1.56 varSsh

Type:	real
Units:	m
Dimension:	nCells Time
Persistence:	persistent
Default Streams:	Output
Location in code:	domain % blocklist % state % time_levs(:) % state % varSsh

Table B.56: varSsh: variance of sea surface height

B.1.57 avgNormalVelocityZonal

Type:	real
Units:	$m s^{-1}$
Dimension:	nVertLevels nCells Time
Persistence:	persistent

Default Streams:	Output
Location in code:	domain % blocklist % state % time_levs(:) % state % avgNormalVelocityZonal

Table B.57: avgNormalVelocityZonal: time-averaged velocity in the eastward direction

B.1.58 avgNormalVelocityMeridional

Type:	real
Units:	$m s^{-1}$
Dimension:	nVertLevels nCells Time
Persistence:	persistent
Default Streams:	Output
Location in code:	domain % blocklist % state % time_levs(:) % state % avgNormalVelocityMeridional

Table B.58: avgNormalVelocityMeridional: time-averaged velocity in the northward direction

B.1.59 varNormalVelocityZonal

Type:	real
Units:	$m s^{-1}$
Dimension:	nVertLevels nCells Time
Persistence:	persistent
Default Streams:	Output
Location in code:	domain % blocklist % state % time_levs(:) % state % varNormalVelocityZonal

Table B.59: varNormalVelocityZonal: variance of velocity in the eastward direction

B.1.60 varNormalVelocityMeridional

Type:	real
Units:	$m s^{-1}$
Dimension:	nVertLevels nCells Time
Persistence:	persistent
Default Streams:	Output
Location in code:	domain % blocklist % state % time_levs(:) % state % varNormalVelocityMeridional

Table B.60: varNormalVelocityMeridional: variance of velocity in the northward direction

B.1.61 avgNormalVelocity

Type:	real
Units:	$m s^{-1}$
Dimension:	nVertLevels nEdges Time
Persistence:	persistent
Default Streams:	Output
Location in code:	domain % blocklist % state % time_levs(:) % state % avgNormalVelocity

Table B.61: avgNormalVelocity: time-averaged velocity, normal to cell edge

B.1.62 varNormalVelocity

Type:	real
Units:	$m s^{-1}$
Dimension:	nVertLevels nEdges Time
Persistence:	persistent
Default Streams:	Output
Location in code:	domain % blocklist % state % time_levs(:) % state % varNormalVelocity

Table B.62: varNormalVelocity: variance of velocity, normal to cell edge

B.1.63 avgVertVelocityTop

Type:	real
Units:	$m s^{-1}$
Dimension:	nVertLevelsP1 nCells Time
Persistence:	persistent
Default Streams:	Output
Location in code:	domain % blocklist % state % time_levs(:) % state % avgVertVelocityTop

Table B.63: avgVertVelocityTop: time-averaged vertical velocity at top of cell

B.2 mesh

B.2.1 latCell

Type:	real
Units:	<i>radians</i>
Dimension:	nCells
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % latCell

Table B.64: latCell: Latitude location of cell centers in radians.

B.2.2 lonCell

Type:	real
Units:	<i>radians</i>
Dimension:	nCells
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % lonCell

Table B.65: lonCell: Longitude location of cell centers in radians.

B.2.3 xCell

Type:	real
Units:	<i>unitless</i>
Dimension:	nCells
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % xCell

Table B.66: xCell: X Coordinate in cartesian space of cell centers.

B.2.4 yCell

Type:	real
Units:	<i>unitless</i>
Dimension:	nCells

Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % yCell

Table B.67: yCell: Y Coordinate in cartesian space of cell centers.

B.2.5 zCell

Type:	real
Units:	<i>unitless</i>
Dimension:	nCells
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % zCell

Table B.68: zCell: Z Coordinate in cartesian space of cell centers.

B.2.6 indexToCellID

Type:	integer
Units:	<i>unitless</i>
Dimension:	nCells
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % indexToCellID

Table B.69: indexToCellID: List of global cell IDs.

B.2.7 latEdge

Type:	real
Units:	<i>radians</i>
Dimension:	nEdges
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % latEdge

Table B.70: latEdge: Latitude location of edge midpoints in radians.

B.2.8 lonEdge

Type:	real
Units:	<i>radians</i>
Dimension:	nEdges
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % lonEdge

Table B.71: lonEdge: Longitude location of edge midpoints in radians.

B.2.9 xEdge

Type:	real
Units:	<i>unitless</i>
Dimension:	nEdges
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % xEdge

Table B.72: xEdge: X Coordinate in cartesian space of edge midpoints.

B.2.10 yEdge

Type:	real
Units:	<i>unitless</i>
Dimension:	nEdges
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % yEdge

Table B.73: yEdge: Y Coordinate in cartesian space of edge midpoints.

B.2.11 zEdge

Type:	real
Units:	<i>unitless</i>
Dimension:	nEdges
Persistence:	persistent
Default Streams:	Input Restart Output

Location in code:	domain % blocklist % mesh % zEdge
-------------------	-----------------------------------

Table B.74: zEdge: Z Coordinate in cartesian space of edge midpoints.

B.2.12 indexToEdgeID

Type:	integer
Units:	<i>unitless</i>
Dimension:	nEdges
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % indexToEdgeID

Table B.75: indexToEdgeID: List of global edge IDs.

B.2.13 latVertex

Type:	real
Units:	<i>radians</i>
Dimension:	nVertices
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % latVertex

Table B.76: latVertex: Latitude location of vertices in radians.

B.2.14 lonVertex

Type:	real
Units:	<i>radians</i>
Dimension:	nVertices
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % lonVertex

Table B.77: lonVertex: Longitude location of vertices in radians.

B.2.15 xVertex

Type:	real
Units:	<i>unitless</i>
Dimension:	nVertices
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % xVertex

Table B.78: xVertex: X Coordinate in cartesian space of vertices.

B.2.16 **yVertex**

Type:	real
Units:	<i>unitless</i>
Dimension:	nVertices
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % yVertex

Table B.79: yVertex: Y Coordinate in cartesian space of vertices.

B.2.17 **zVertex**

Type:	real
Units:	<i>unitless</i>
Dimension:	nVertices
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % zVertex

Table B.80: zVertex: Z Coordinate in cartesian space of vertices.

B.2.18 **indexToVertexID**

Type:	integer
Units:	<i>unitless</i>
Dimension:	nVertices
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % indexToVertexID

Table B.81: indexToVertexID: List of global vertex IDs.

B.2.19 `meshDensity`

Type:	real
Units:	<i>unitless</i>
Dimension:	nCells
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % meshDensity

Table B.82: `meshDensity`: Value of density function used to generate a particular mesh at cell centers.

B.2.20 `meshScalingDel2`

Type:	real
Units:	<i>unitless</i>
Dimension:	nEdges
Persistence:	persistent
Default Streams:	Restart Output
Location in code:	domain % blocklist % mesh % meshScalingDel2

Table B.83: `meshScalingDel2`: Coefficient to Laplacian mixing terms in momentum and tracer equations, so that viscosity and diffusion scale with mesh.

B.2.21 `meshScalingDel4`

Type:	real
Units:	<i>unitless</i>
Dimension:	nEdges
Persistence:	persistent
Default Streams:	Restart Output
Location in code:	domain % blocklist % mesh % meshScalingDel4

Table B.84: `meshScalingDel4`: Coefficient to biharmonic mixing terms in momentum and tracer equations, so that biharmonic viscosity and diffusion coefficients scale with mesh.

B.2.22 `meshScaling`

Type:	real
Units:	<i>unitless</i>
Dimension:	nEdges
Persistence:	persistent
Default Streams:	Restart Output
Location in code:	domain % blocklist % mesh % meshScaling

Table B.85: meshScaling: Coefficient used for mesh scaling, such as the Leith parameter.

B.2.23 [cellsOnEdge](#)

Type:	integer
Units:	<i>unitless</i>
Dimension:	TWO nEdges
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % cellsOnEdge

Table B.86: cellsOnEdge: List of cells that straddle each edge.

B.2.24 [nEdgesOnCell](#)

Type:	integer
Units:	<i>unitless</i>
Dimension:	nCells
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % nEdgesOnCell

Table B.87: nEdgesOnCell: Number of edges that border each cell.

B.2.25 [nEdgesOnEdge](#)

Type:	integer
Units:	<i>unitless</i>
Dimension:	nEdges
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % nEdgesOnEdge

Table B.88: nEdgesOnEdge: Number of edges that surround each of the cells that straddle each edge. These edges are used to reconstruct the tangential velocities.

B.2.26 `edgesOnCell`

Type:	integer
Units:	<i>unitless</i>
Dimension:	maxEdges nCells
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % edgesOnCell

Table B.89: edgesOnCell: List of edges that border each cell.

B.2.27 `edgesOnEdge`

Type:	integer
Units:	<i>unitless</i>
Dimension:	maxEdges2 nEdges
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % edgesOnEdge

Table B.90: edgesOnEdge: List of edges that border each of the cells that straddle each edge.

B.2.28 `weightsOnEdge`

Type:	real
Units:	<i>unitless</i>
Dimension:	maxEdges2 nEdges
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % weightsOnEdge

Table B.91: weightsOnEdge: Reconstruction weights associated with each of the edgesOnEdge.

B.2.29 dvEdge

Type:	real
Units:	m
Dimension:	nEdges
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % dvEdge

Table B.92: dvEdge: Length of each edge, computed as the distance between verticesOnEdge.

B.2.30 dcEdge

Type:	real
Units:	m
Dimension:	nEdges
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % dcEdge

Table B.93: dcEdge: Length of each edge, computed as the distance between cellsOnEdge.

B.2.31 angleEdge

Type:	real
Units:	<i>radians</i>
Dimension:	nEdges
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % angleEdge

Table B.94: angleEdge: Angle the edge normal makes with local eastward direction.

B.2.32 areaCell

Type:	real
Units:	m^2
Dimension:	nCells

Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % areaCell

Table B.95: areaCell: Area of each cell in the primary grid.

B.2.33 [areaTriangle](#)

Type:	real
Units:	m^2
Dimension:	nVertices
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % areaTriangle

Table B.96: areaTriangle: Area of each cell (triangle) in the dual grid.

B.2.34 [edgeNormalVectors](#)

Type:	real
Units:	<i>unitless</i>
Dimension:	R3 nEdges
Persistence:	persistent
Default Streams:	Output
Location in code:	domain % blocklist % mesh % edgeNormalVectors

Table B.97: edgeNormalVectors: Normal vector defined at an edge.

B.2.35 [localVerticalUnitVectors](#)

Type:	real
Units:	<i>unitless</i>
Dimension:	R3 nCells
Persistence:	persistent
Default Streams:	Output
Location in code:	domain % blocklist % mesh % localVerticalUnitVectors

Table B.98: localVerticalUnitVectors: Unit surface normal vectors defined at cell centers.

B.2.36 `cellTangentPlane`

Type:	real
Units:	<i>unitless</i>
Dimension:	R3 TWO nCells
Persistence:	persistent
Default Streams:	Output
Location in code:	domain % blocklist % mesh % cellTangentPlane

Table B.99: `cellTangentPlane`: The two vectors that define a tangent plane at a cell center.

B.2.37 `cellsOnCell`

Type:	integer
Units:	<i>unitless</i>
Dimension:	maxEdges nCells
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % cellsOnCell

Table B.100: `cellsOnCell`: List of cells that neighbor each cell.

B.2.38 `verticesOnCell`

Type:	integer
Units:	<i>unitless</i>
Dimension:	maxEdges nCells
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % verticesOnCell

Table B.101: `verticesOnCell`: List of vertices that border each cell.

B.2.39 `verticesOnEdge`

Type:	integer
Units:	<i>unitless</i>
Dimension:	TWO nEdges
Persistence:	persistent

Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % verticesOnEdge

Table B.102: verticesOnEdge: List of vertices that straddle each edge.

B.2.40 edgesOnVertex

Type:	integer
Units:	<i>unitless</i>
Dimension:	vertexDegree nVertices
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % edgesOnVertex

Table B.103: edgesOnVertex: List of edges that share a vertex as an endpoint.

B.2.41 cellsOnVertex

Type:	integer
Units:	<i>unitless</i>
Dimension:	vertexDegree nVertices
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % cellsOnVertex

Table B.104: cellsOnVertex: List of cells that share a vertex.

B.2.42 kiteAreasOnVertex

Type:	real
Units:	m^2
Dimension:	vertexDegree nVertices
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % kiteAreasOnVertex

Table B.105: kiteAreasOnVertex: Area of the portions of each dual cell that are part of each cellsOnVertex.

B.2.43 **fEdge**

Type:	real
Units:	s^{-1}
Dimension:	nEdges
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % fEdge

Table B.106: fEdge: Coriolis parameter at edges.

B.2.44 **fVertex**

Type:	real
Units:	s^{-1}
Dimension:	nVertices
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % fVertex

Table B.107: fVertex: Coriolis parameter at vertices.

B.2.45 **bottomDepth**

Type:	real
Units:	m
Dimension:	nCells
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % bottomDepth

Table B.108: bottomDepth: Depth of the bottom of the ocean. Given as a positive distance from sea level.

B.2.46 **deriv_two**

Type:	real
Units:	m^{-2}
Dimension:	maxEdges2 TWO nEdges
Persistence:	persistent

Default Streams:	None
Location in code:	domain % blocklist % mesh % deriv_two

Table B.109: deriv_two: Value of the second derivative of the polynomial used for reconstruction of cell center quantities at edges.

B.2.47 adv_coefs

Type:	real
Units:	m
Dimension:	nAdvectionCells nEdges
Persistence:	persistent
Default Streams:	None
Location in code:	domain % blocklist % mesh % adv_coefs

Table B.110: adv_coefs: Weighting coefficients used for reconstruction of cell center quantities at edges. Used in advection routines.

B.2.48 adv_coefs_2nd

Type:	real
Units:	m
Dimension:	nAdvectionCells nEdges
Persistence:	persistent
Default Streams:	None
Location in code:	domain % blocklist % mesh % adv_coefs_2nd

Table B.111: adv_coefs_2nd: Weighting coefficients used for reconstruction of cell center quantities at edges. Used in advection routines.

B.2.49 adv_coefs_3rd

Type:	real
Units:	m
Dimension:	nAdvectionCells nEdges
Persistence:	persistent
Default Streams:	None
Location in code:	domain % blocklist % mesh % adv_coefs_3rd

Table B.112: adv_coefs_3rd: Weighting coefficients used for reconstruction of cell center quantities at edges. Used in advection routines.

B.2.50 advCellsForEdge

Type:	integer
Units:	<i>unitless</i>
Dimension:	nAdvectionCells nEdges
Persistence:	persistent
Default Streams:	None
Location in code:	domain % blocklist % mesh % advCellsForEdge

Table B.113: advCellsForEdge: List of cells used to reconstruct a cell quantity at an edge.
Used in advection routines.

B.2.51 nAdvCellsForEdge

Type:	integer
Units:	<i>unitless</i>
Dimension:	nEdges
Persistence:	persistent
Default Streams:	None
Location in code:	domain % blocklist % mesh % nAdvCellsForEdge

Table B.114: nAdvCellsForEdge: Number of cells used in reconstruction of cell center quantities at an edge. Used in advection routines.

B.2.52 highOrderAdvectionMask

Type:	integer
Units:	<i>unitless</i>
Dimension:	nVertLevels nEdges
Persistence:	persistent
Default Streams:	None
Location in code:	domain % blocklist % mesh % highOrderAdvectionMask

Table B.115: highOrderAdvectionMask: Mask for high order advection. Values are 1 if high order is used, and 0 if not.

B.2.53 lowOrderAdvectionMask

Type:	integer
Units:	<i>unitless</i>
Dimension:	nVertLevels nEdges
Persistence:	persistent
Default Streams:	None
Location in code:	domain % blocklist % mesh % lowOrderAdvectionMask

Table B.116: lowOrderAdvectionMask: Mask for low order advection. Values are 1 if low order is used, and 0 if not.

B.2.54 [defc_a](#)

Type:	real
Units:	m^{-1}
Dimension:	maxEdges nCells
Persistence:	persistent
Default Streams:	None
Location in code:	domain % blocklist % mesh % defc_a

Table B.117: defc_a: Variable used with advection setup to compute advection coefficients. Deformation weight coefficients.

B.2.55 [defc_b](#)

Type:	real
Units:	m^{-1}
Dimension:	maxEdges nCells
Persistence:	persistent
Default Streams:	None
Location in code:	domain % blocklist % mesh % defc_b

Table B.118: defc_b: Variable used with advection setup to compute advection coefficients. Deformation weight coefficients.

B.2.56 [coeffs_reconstruct](#)

Type:	real
Units:	<i>unitless</i>
Dimension:	R3 maxEdges nCells
Persistence:	persistent

Default Streams:	None
Location in code:	domain % blocklist % mesh % coeffs_reconstruct

Table B.119: coeffs_reconstruct: Coefficients to reconstruct velocity vectors at cells centers.

B.2.57 `maxLevelCell`

Type:	integer
Units:	<i>unitless</i>
Dimension:	nCells
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % maxLevelCell

Table B.120: maxLevelCell: Index to the last active ocean cell in each column.

B.2.58 `maxLevelEdgeTop`

Type:	integer
Units:	<i>unitless</i>
Dimension:	nEdges
Persistence:	persistent
Default Streams:	None
Location in code:	domain % blocklist % mesh % maxLevelEdgeTop

Table B.121: maxLevelEdgeTop: Index to the last edge in a column with active ocean cells on both sides of it.

B.2.59 `maxLevelEdgeBot`

Type:	integer
Units:	<i>unitless</i>
Dimension:	nEdges
Persistence:	persistent
Default Streams:	None
Location in code:	domain % blocklist % mesh % maxLevelEdgeBot

Table B.122: maxLevelEdgeBot: Index to the last edge in a column with at least one active ocean cell on either side of it.

B.2.60 `maxLevelVertexTop`

Type:	integer
Units:	<i>unitless</i>
Dimension:	nVertices
Persistence:	persistent
Default Streams:	None
Location in code:	domain % blocklist % mesh % maxLevelVertexTop

Table B.123: `maxLevelVertexTop`: Index to the last vertex in a column with all active cells around it.

B.2.61 `maxLevelVertexBot`

Type:	integer
Units:	<i>unitless</i>
Dimension:	nVertices
Persistence:	persistent
Default Streams:	None
Location in code:	domain % blocklist % mesh % maxLevelVertexBot

Table B.124: `maxLevelVertexBot`: Index to the last vertex in a column with at least one active ocean cell around it.

B.2.62 `refBottomDepth`

Type:	real
Units:	<i>m</i>
Dimension:	nVertLevels
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % refBottomDepth

Table B.125: `refBottomDepth`: Reference depth of ocean for each vertical level. Used in 'z-level' type runs.

B.2.63 `refBottomDepthTopOfCell`

Type:	real
Units:	<i>m</i>
Dimension:	nVertLevelsP1
Persistence:	persistent
Default Streams:	None
Location in code:	domain % blocklist % mesh % refBottomDepthTopOfCell

Table B.126: refBottomDepthTopOfCell: Reference depth of ocean for each vertical interface. Used in 'z-level' type runs.

B.2.64 **vertCoordMovementWeights**

Type:	real
Units:	<i>unitless</i>
Dimension:	nVertLevels
Persistence:	persistent
Default Streams:	Input Restart Output
Location in code:	domain % blocklist % mesh % vertCoordMovementWeights

Table B.127: vertCoordMovementWeights: Weights used for distribution of sea surface height perturbations through multiple vertical levels.

B.2.65 **boundaryEdge**

Type:	integer
Units:	<i>unitless</i>
Dimension:	nVertLevels nEdges
Persistence:	persistent
Default Streams:	None
Location in code:	domain % blocklist % mesh % boundaryEdge

Table B.128: boundaryEdge: Mask for determining boundary edges. A boundary edge has only one active ocean cell neighboring it.

B.2.66 **boundaryVertex**

Type:	integer
Units:	<i>unitless</i>
Dimension:	nVertLevels nVertices
Persistence:	persistent

Default Streams:	None
Location in code:	domain % blocklist % mesh % boundaryVertex

Table B.129: boundaryVertex: Mask for determining boundary vertices. A boundary vertex has at least one inactive cell neighboring it.

B.2.67 [boundaryCell](#)

Type:	integer
Units:	<i>unitless</i>
Dimension:	nVertLevels nCells
Persistence:	persistent
Default Streams:	None
Location in code:	domain % blocklist % mesh % boundaryCell

Table B.130: boundaryCell: Mask for determining boundary cells. A boundary cell has at least one inactive cell neighboring it.

B.2.68 [edgeMask](#)

Type:	integer
Units:	<i>unitless</i>
Dimension:	nVertLevels nEdges
Persistence:	persistent
Default Streams:	Output
Location in code:	domain % blocklist % mesh % edgeMask

Table B.131: edgeMask: Mask on edges that determines if computations should be done on edge.

B.2.69 [vertexMask](#)

Type:	integer
Units:	<i>unitless</i>
Dimension:	nVertLevels nVertices
Persistence:	persistent
Default Streams:	Output
Location in code:	domain % blocklist % mesh % vertexMask

Table B.132: vertexMask: Mask on vertices that determines if computations should be done on vertice.

B.2.70 `cellMask`

Type:	integer
Units:	<i>unitless</i>
Dimension:	nVertLevels nCells
Persistence:	persistent
Default Streams:	Output
Location in code:	domain % blocklist % mesh % cellMask

Table B.133: `cellMask`: Mask on cells that determines if computations should be done on cell.

B.2.71 `normalVelocityForcing`

Type:	real
Units:	$N \ m^{-2}$
Dimension:	nVertLevels nEdges
Persistence:	persistent
Default Streams:	Input Restart
Location in code:	domain % blocklist % mesh % normalVelocityForcing

Table B.134: `normalVelocityForcing`: Velocity forcing field. Defines a forcing at an edge.

B.2.72 `temperatureRestore`

Type:	real
Units:	$^{\circ} C$
Dimension:	nCells
Persistence:	persistent
Default Streams:	Input Restart
Location in code:	domain % blocklist % mesh % temperatureRestore

Table B.135: `temperatureRestore`: Temperature restoring field, for restoring temperature at the surface.

B.2.73 `salinityRestore`

Type:	real
-------	------

Units:	<i>PSU</i>
Dimension:	nCells
Persistence:	persistent
Default Streams:	Input Restart
Location in code:	domain % blocklist % mesh % salinityRestore

Table B.136: salinityRestore: Salinity restoring field, for restoring salinity at the surface.

B.2.74 [windStressMonthly](#)

Type:	real
Units:	$N\ m^{-2}$
Dimension:	nMonths nEdges
Persistence:	persistent
Default Streams:	Input Restart
Location in code:	domain % blocklist % mesh % windStressMonthly

Table B.137: windStressMonthly: Monthly wind stress field, defined at the surface for use in monthly forcing.

B.2.75 [temperatureRestoreMonthly](#)

Type:	real
Units:	$^{\circ}C$
Dimension:	nMonths nCells
Persistence:	persistent
Default Streams:	Input Restart
Location in code:	domain % blocklist % mesh % temperatureRestoreMonthly

Table B.138: temperatureRestoreMonthly: Monthly temperature restorying field, defined at the surface for use in monthly forcing.

B.2.76 [salinityRestoreMonthly](#)

Type:	real
Units:	<i>PSU</i>
Dimension:	nMonths nCells
Persistence:	persistent
Default Streams:	Input Restart
Location in code:	domain % blocklist % mesh % salinityRestoreMonthly

Table B.139: `salinityRestoreMonthly`: Monthly salinity resotring field, defined at the surface, for use in monthly forcing.

B.2.77 `edgeSignOnCell`

Type:	integer
Units:	<i>unitless</i>
Dimension:	maxEdges nCells
Persistence:	persistent
Default Streams:	None
Location in code:	domain % blocklist % mesh % edgeSignOnCell

Table B.140: `edgeSignOnCell`: Sign of edge contributions to a cell for each edge on cell.
Used for bit-reproducible loops. Represents directionality of vector connecting cells.

B.2.78 `edgeSignOnVertex`

Type:	integer
Units:	<i>unitless</i>
Dimension:	maxEdges nVertices
Persistence:	persistent
Default Streams:	None
Location in code:	domain % blocklist % mesh % edgeSignOnVertex

Table B.141: `edgeSignOnVertex`: Sign of edge contributions to a vertex for each edge on vertex. Used for bit-reproducible loops. Represents directionality of vector connecting vertices.

B.2.79 `kiteIndexOnCell`

Type:	integer
Units:	<i>unitless</i>
Dimension:	maxEdges nCells
Persistence:	persistent
Default Streams:	None
Location in code:	domain % blocklist % mesh % kiteIndexOnCell

Table B.142: `kiteIndexOnCell`: Index of kite in dual grid, based on `verticesOnCell`.

B.2.80 seaSurfacePressure

Type:	real
Units:	Pa
Dimension:	nCells Time
Persistence:	persistent
Default Streams:	Input Restart
Location in code:	domain % blocklist % mesh % seaSurfacePressure

Table B.143: seaSurfacePressure: Pressure defined at the sea surface.

B.3 tend

B.3.1 tend_temperature

Type:	real
Units:	$K s^{-1}$
Dimension:	nVertLevels nCells Time
Persistence:	persistent
Default Streams:	None
Index in tracers Array:	domain % blocklist % tend % index_temperature
Location in code:	domain % blocklist % tend % tracers
Array Group:	dynamics

Table B.144: tend_temperature: time tendency of potential temperature

B.3.2 tend_salinity

Type:	real
Units:	$PSU s^{-1}$
Dimension:	nVertLevels nCells Time
Persistence:	persistent
Default Streams:	None
Index in tracers Array:	domain % blocklist % tend % index_salinity
Location in code:	domain % blocklist % tend % tracers
Array Group:	dynamics

Table B.145: tend_salinity: time tendency of salinity measured as change in practical salinity units per second

B.3.3 `tend_normalVelocity`

Type:	real
Units:	$m s^{-2}$
Dimension:	nVertLevels nEdges Time
Persistence:	persistent
Default Streams:	None
Location in code:	domain % blocklist % tend % normalVelocity

Table B.146: `tend_normalVelocity`: time tendency of normal component of velocity

B.3.4 `tend_layerThickness`

Type:	real
Units:	$m s^{-1}$
Dimension:	nVertLevels nCells Time
Persistence:	persistent
Default Streams:	None
Location in code:	domain % blocklist % tend % layerThickness

Table B.147: `tend_layerThickness`: time tendency of layer thickness

B.3.5 `tend_ssh`

Type:	real
Units:	$m s^{-1}$
Dimension:	nCells Time
Persistence:	persistent
Default Streams:	None
Location in code:	domain % blocklist % tend % ssh

Table B.148: `tend_ssh`: time tendency of sea-surface height

B.4 diagnostics

B.4.1 `RiTopOfCell`

Type:	real
-------	------

Units:	<i>nondimensional</i>
Dimension:	nVertLevelsP1 nCells Time
Persistence:	persistent
Default Streams:	None
Location in code:	domain % blocklist % diagnostics % RiTopOfCell

Table B.149: RiTopOfCell: gradient Richardson number defined at the center (horizontally) and top (vertically)

B.4.2 **RiTopOfEdge**

Type:	real
Units:	<i>nondimensional</i>
Dimension:	nVertLevelsP1 nEdges Time
Persistence:	persistent
Default Streams:	None
Location in code:	domain % blocklist % diagnostics % RiTopOfEdge

Table B.150: RiTopOfEdge: gradient Richardson number defined at the edge (horizontally) and top (vertically)

B.4.3 **vertViscTopOfEdge**

Type:	real
Units:	$m^2 s^{-1}$
Dimension:	nVertLevelsP1 nEdges Time
Persistence:	persistent
Default Streams:	None
Location in code:	domain % blocklist % diagnostics % vertViscTopOfEdge

Table B.151: vertViscTopOfEdge: vertical viscosity defined at the edge (horizontally) and top (vertically)

B.4.4 **vertDiffTopOfCell**

Type:	real
Units:	$m^2 s^{-1}$
Dimension:	nVertLevelsP1 nCells Time
Persistence:	persistent
Default Streams:	None

Location in code:	domain % blocklist % diagnostics % vertDiffTopOfCell
-------------------	--

Table B.152: `vertDiffTopOfCell`: vertical diffusion defined at the edge (horizontally) and top (vertically)

B.4.5 `windStressZonal`

Type:	real
Units:	$N\ m^{-2}$
Dimension:	nCells Time
Persistence:	persistent
Default Streams:	Output
Location in code:	domain % blocklist % diagnostics % windStressZonal

Table B.153: `windStressZonal`: reconstructed surface wind stress in the eastward direction.

B.4.6 `windStressMeridional`

Type:	real
Units:	$N\ m^{-2}$
Dimension:	nCells Time
Persistence:	persistent
Default Streams:	Output
Location in code:	domain % blocklist % diagnostics % windStressMeridional

Table B.154: `windStressMeridional`: reconstructed surface wind stress in the northward direction.

B.5 `scratch`

B.5.1 `normalVelocityForcingX`

Type:	real
Units:	$N\ m^{-2}$
Dimension:	nVertLevels nCells Time
Persistence:	persistent
Default Streams:	None
Location in code:	domain % blocklist % scratch % normalVelocityForcingX

Table B.155: `normalVelocityForcingX`: reconstructed wind stress in the x-direction (cartesian)

B.5.2 `normalVelocityForcingY`

Type:	real
Units:	$N\ m^{-2}$
Dimension:	nVertLevels nCells Time
Persistence:	persistent
Default Streams:	None
Location in code:	domain % blocklist % scratch % normalVelocityForcingY

Table B.156: `normalVelocityForcingY`: reconstructed wind stress in the y-direction (cartesian)

B.5.3 `normalVelocityForcingZ`

Type:	real
Units:	$N\ m^{-2}$
Dimension:	nVertLevels nCells Time
Persistence:	persistent
Default Streams:	None
Location in code:	domain % blocklist % scratch % normalVelocityForcingZ

Table B.157: `normalVelocityForcingZ`: reconstructed wind stress in the z-direction (cartesian)

B.5.4 `normalVelocityForcingZonal`

Type:	real
Units:	$N\ m^{-2}$
Dimension:	nVertLevels nCells Time
Persistence:	persistent
Default Streams:	None
Location in code:	domain % blocklist % scratch % normalVelocityForcingZonal

Table B.158: `normalVelocityForcingZonal`: reconstructed wind stress in the eastward direction

B.5.5 `normalVelocityForcingMeridional`

Type:	real
Units:	$N \ m^{-2}$
Dimension:	nVertLevels nCells Time
Persistence:	persistent
Default Streams:	None
Location in code:	domain % blocklist % scratch % normalVelocityForcing-Meridional

Table B.159: `normalVelocityForcingMeridional`: reconstructed wind stress in the northward direction

B.5.6 `vorticityGradientTangentialComponent`

Type:	real
Units:	$s^{-1} \ m^{-1}$
Dimension:	nVertLevels nEdges Time
Persistence:	persistent
Default Streams:	None
Location in code:	domain % blocklist % scratch % vorticityGradientTangential-Component

Table B.160: `vorticityGradientTangentialComponent`: gradient of vorticity in the tangent direction (positive points from vertex1 to vertex2)

B.5.7 `vorticityGradientNormalComponent`

Type:	real
Units:	$s^{-1} \ m^{-1}$
Dimension:	nVertLevels nEdges Time
Persistence:	persistent
Default Streams:	None
Location in code:	domain % blocklist % scratch % vorticityGradientNormal-Component

Table B.161: `vorticityGradientNormalComponent`: gradient of vorticity in the normal direction (positive points from cell1 to cell2)

B.5.8 **normalizedRelativeVorticityVertex**

Type:	real
Units:	s^{-1}
Dimension:	nVertLevels nVertices Time
Persistence:	persistent
Default Streams:	None
Location in code:	domain % blocklist % scratch % normalizedRelativeVorticityVertex

Table B.162: normalizedRelativeVorticityVertex: curl of horizontal velocity divided by layer thickness, defined at vertices

B.5.9 **normalizedPlanetaryVorticityVertex**

Type:	real
Units:	s^{-1}
Dimension:	nVertLevels nVertices Time
Persistence:	persistent
Default Streams:	None
Location in code:	domain % blocklist % scratch % normalizedPlanetaryVorticityVertex

Table B.163: normalizedPlanetaryVorticityVertex: earth's rotational rate (Coriolis parameter, f) divided by layer thickness, defined at vertices

B.5.10 **kineticEnergyVertex**

Type:	real
Units:	$m^2 s^{-2}$
Dimension:	nVertLevels nVertices Time
Persistence:	persistent
Default Streams:	None
Location in code:	domain % blocklist % scratch % kineticEnergyVertex

Table B.164: kineticEnergyVertex: kinetic energy of horizontal velocity defined at vertices

B.5.11 **kineticEnergyVertexOnCells**

Type:	real
-------	------

Units:	$m^2 s^{-2}$
Dimension:	nVertLevels nCells Time
Persistence:	persistent
Default Streams:	None
Location in code:	domain % blocklist % scratch % kineticEnergyVertexOnCells

Table B.165: kineticEnergyVertexOnCells: kinetic energy of horizontal velocity defined at vertices